GOLDEN FLUTES & GREAT ESCAPES

How to Write Adventure Games

Delton T. Horn

dilithium Press

Golden Flutes and Great Escapes

Golden Flutes and Great Escapes

How to Write Adventure Games

Delton T. Horn

dilithium Press
Beaverton, Oregon

10 9 8 7 6 5 4 3 2 1

Library of Congress Cataloging in Publication Data

Horn, Delton T.
Golden flutes and great escapes.

Includes index.
1. Computer games. 2. TRS-80 (Computer)—Programming.
I. Title.
GVI469.2.H674 1984 794.8'2 83-18994
ISBN 0-88056-089-4

Cover: Vernon G. Groff

Printed in the United States of America

dilithium Press
Suite 151
8285 S.W. Nimbus
Beaverton, Oregon 97005

An Important Note

The publisher and the authors have made every effort to ensure that the computer programs and programming information in this publication are accurate and complete. However, this publication is prepared for general readership, and neither the publisher nor the authors have any knowledge about or ability to control any third party's use of the programs and programming information. There is no warranty or representation by either the publisher or the authors that the programs or programming information in this book will enable the reader or user to achieve any particular result.

Preface

Adventure games add a playful dimension to owning a home computer. This book gives examples of action-filled games that bring you hours of fun; soon you will be writing your own original games for fun—and profit.

These games were written on a 48K TRS-80 model III microcomputer but can be easily adapted to any comparable home computer.

While you need not be a programming whiz to understand and enjoy these programs, beginners will want to become familiar with BASIC programming.

Table of Contents

Chapter 1

Discovering Adventure Games

> You move cautiously down a long, dark corridor beneath the castle of Nembuzur. You notice that your torch is beginning to burn a little low. You'll have to find a new one soon.
>
> At the end of the hall, you come to three closed doors. After a moment's hesitation, you open the third door. A saber-toothed tiger leaps out at you and injures your left arm. Quickly you draw your magic sword. . . .

Is this a dream? No, you are a character in a computer adventure game, a rapidly growing American pasttime. While so much of our entertainment these days is passive—television, movies, books, spectator sports—the more active video games have caught on in a big way. Most of these games are skill-oriented. They test your reflexes but not your creativity and imagination. You either shoot down the aliens, or they shoot you down. There are very few ways to vary the flow of a game.

Sure, it often takes considerable intelligence to aim and time your shots and can be lots of fun. But they are still mentally passive activities.

Traditional board games like backgammon or chess engage our brains more with logical thinking and strategy. They encourage creative, active thinking. Even so, things are still pretty cut and dried. How can the adventures of a checker fully capture your imagination?

A good adventure game in which you act out intrigue is far more engaging. You determine the hero's adventures until he either triumphs or is killed. You can interact with a story and write it as you go. You participate fully in the creation of the fantasy and, once hooked, might stay with the game for hours. (One software manufacturer calls its series of adventure programs "Interactive Fiction.")

Adventure games are ideally suited for computers. Non-computerized adventure games have been around for quite awhile but tend to be rather awkward and complicated to play. Thus they appeal to a relatively small group of enthusiasts who are willing to tackle instruction books as long as novels. Many of these games require an extra inactive player (often called the Loremaster), who sets up the situation and plants the various monsters and treasures to surprise the main player (or players).

Note that most adventure games, computerized or not, are designed for a single hero/player. Too many creative participants can spoil the fantasy.

In computerized games, you still have to learn the basic rules, but you can simply feel your way through the details by trial, error, and logic. The computer simply won't accept an invalid move and will often prompt you for the correct move. So you do not need an extensive rule book. Figuring out the rules and what you can do with various moves can be part of the fun.

The computer also keeps track of all the necessary details—your score, the number of lurking monsters, your character's current health, etc. Computerized adventure games also eliminate complex score sheets. Of the many fine adventure game programs commercially available, the best can be a lot of fun but most suffer from significant disadvantages. One disadvantage which they all suffer from is fixed details. That is, the game is always the same with no random elements. The sack of gold is always hidden behind the bird-god statue. Solving the puzzle can be fun, getting a little closer to the final goal each time you play. But once you know the solution, there is little point in playing anymore, and the program becomes a dust collector. After you know where everything is, it's not much fun.

Many commercially marketed games are in machine language, which speeds up the workings of the program but does not let you customize the game in any way. Many of these programs are software protected (to prevent copying), and you have no access to the program code at all.

This book hopes to provide an alternative. Good adventure games are complex and, therefore, rather tricky to program. But with some fundamental rules and a few tricks that are outlined in the following chapters, you should be able to program your own fantasies. They are fun to write and fun to play, especially if you throw in plenty of randomness. And it's fun to swap programs with imaginative friends.

Moreover, there is a thriving market for good pre-written adventure games, so you can sell your game for a little extra cash after you've had your fun.

GETTING STARTED

All you need to get started in writing adventure game programs is a computer, your imagination and, perhaps, a little deviousness.

A printer is highly desirable so that you can get hard copies of your incomplete programs as you go along. It is far easier to spot errors, that will inevitably occur, in a hard copy than on a CRT screen. Without a printer, the job will be more tedious and time-consuming.

You will also need plenty of memory space. Rarely is 4K or 8K enough for advanture game programs unless they are quite simple and written in machine or assembly langugae. In this book, you work exclusively with BASIC. A little bit of inefficiency and slower calculations are acceptable. Of course, if you are an advanced programmer, you could convert the techniques discussed into assembly language.

For working in BASIC, 16K is really the bare minimum. Of the games described in the following chapters, TREASURE HUNT and THE GOLDEN FLUTE could be fit into 16K; THE GREAT ESCAPE requires about 32K (for the complete version); and the full version of MARS takes close to 48K.

Of course, you can strip any of these games down to a smaller memory size if you eliminate certain features. Each game includes suggestions for reducing memory requirements.

All of the programs and examples in this book were written on a 48K TRS-80 Model III. By comparing these commands with your computer's owner's manual, you should be able to translate them to your machine. Unusual, esoteric commands are avoided. When you write your own games, feel free to use all of the features and programming tricks in your computer's repertoire.

You don't need to be a whiz, but you should have some experience on BASIC fundamentals to benefit from this book.

You could simply type in the complete programs included here and in fact, are invited to do so. Right there you've got a bargain as most single adventure game programs cost more than this book. However, you will cheat yourself if you stop with the four pre-written games. Creating adventure games is three fourths of the fun.

All techniques and tricks in this book are simply suggestions. There are no absolute right ways to program. Experiment freely. The worst that could possibly happen is you'll be left staring at an error message. Simply locate and correct the error, and try again. Every programmer has to debug. Turn your imagination loose and enjoy the fun.

Chapter 2

Creating a Plot

The first step in writing an adventure game is to determine the kind of adventure you'd like to have. The possibilities are infinite when you use your own interests and imagination.

Explore a haunted house. Rescue a medieval princess from a dungeon. Recreate an historic battle and change the course of history. Build a galactic empire. Do anything you like.

This chapter works up a sample game plot. The actual program is written in the next few chapters.

Since space-oriented games tend to be quite popular, let's write a game about exploring the planet Mars. The name of the game is MARS.

When real-life scientists explore Mars, they're sure to find plenty of scientifically interesting information. But in the adventure game sense, rock collecting doesn't thrill the average person.

Fortunately, you are not limited to the realistic or probable. So imagine there was once a glorious ancient civilization on Mars. It has long since died out, but explore the planet for valuable relics and artifacts in the course of the game.

Already you have a basis for scoring in the game. Each relic we find is worth X number of points. The object of the game is to gather up as many Martian treasures as possible, return them to the space ship, and blast off for Earth.

To create even more interest, seed the planet with worthless items that don't add to the score. In fact, to pick up a piece of junk could cost you Y points to be subtracted from the score.

You can also decide which supplies to take aboard your space ship at the beginning of the game. Supplies shouldn't have any direct influence on the score, but they can greatly influence the explorer's chances of survival.

Arbitrarily, set up an even dozen treasures, 12 pieces of junk, and 12 supply items for a total of 36 items the player might carry. As an extra touch, make it impossible for the player to carry more than 15 items at one time. This will force the player to pick and choose carefully and plan his strategy.

An object hunt isn't too terribly exciting, so let's throw in some monsters. The player will have to deal with each type of monster in a different way to ensure variety. Add some natural obstacles like rivers, mountains, and ravines. While we're at it, weird Martian storms and/or marsquakes (the equivalent of *earth*quake) can liven things up. Our game plot is now pretty well outlined. Things are still rather vague, but the possibilities are beginning to appear. Now fill in some of the details.

At this point you are still not pinning anything down. You can still change your mind about some things. Other ideas may occur to you as we write the program itself. But start tentatively defining things. After all, you have to begin somewhere.

THE GAME FORMAT

First off, consider the basic game structure. Until you determine the format you can't do anything else.

Since MARS is a search game through unknown territory, you should establish a playing board or map in the computer's memory. A 10 by 10 grid (see Figure 2.1) is convenient and gives you 100 areas to explore. Any more could make the game too long and drawn out and take up too much memory space. Any less would make the game too easy.

Figure 2.1 Playing Grid for Mars.

```
   1  2  3  4  5  6  7  8  9  10
A  .  .  .  .  .  .  .  .  .  .
B  .  .  .  .  .  .  .  .  .  .
C  .  .  .  .  X  .  .  .  .  .
D  .  .  .  .  .  .  .  .  .  .
E  .  .  .  .  .  .  .  .  .  .
F  .  .  .  .  .  .  .  .  .  .
G  .  .  .  .  .  .  .  .  .  .
H  .  .  .  .  .  .  .  .  .  .
I  .  .  .  .  .  .  .  .  .  .
J  .  .  .  .  .  .  .  .  .  .
```

X = player's position

You can define four possible moves within the grid. North would be up (from c-5 to b-5), south would be down (from c-5 to d-5), east would be to the right (from c-5 to c-6), and west would be to the left (from c-5 to c-4).

Diagonal moves, like southwest (c-5 to d-4) could also be used but would call for more complicated programming. Since there are a lot of other things we want to put into this program, it is a good idea to conserve memory space by limiting ourselves to the four basic directional moves. If you find you have enough memory space once the program is finished to add diagonal moves, add them at that point. For the time being, however, ignore that particular possiblility.

Whenever you use a playing board map format, make provision for the chance of the player moving outside the defined grid area. For example, if the player is at e-10 and moves east, his location becomes undefined. This could bomb out the program completely, or it may result in weird, unpredictable scores and plays. You must include some form of protection in any game program that uses the map format.

There are four ways to deal with this problem, and you may dream up yet another approach. Feel free to try out any ideas you come up with. The following ideas will give you a start:

1. The simplest way to deal with an off the board move would be to have the computer recognize and refuse such a move. An error message could be then displayed, such as "INVALID MOVE" or "YOU CAN'T GO THAT WAY". The computer would then prompt the player for another move.

2. A more drastic way of dealing with an off the map move is for the bad move to result in instant death, or loss of the game. For example, the computer could print out, "YOU HAVE WANDERED OUTSIDE OF THE KNOWN TERRITORY. YOU WANTER AIMLESSLY ABOUT, HOPELESSLY LOST, UNTIL YOU DIE. GAME OVER."

3. A less drastic penalty would be to randomly relocate the player somewhere within the map grid—possibly up to his neck in trouble. Similarly, in an object gathering game like MARS, you could lose treasures by making an invalid move. You could then hide the forfeited treasures again within the grid.

4. The fourth approach is to simply loop around the map. For example, moving north from a-7 would position the player at j-7. Moving west from h-1 leaves the player at h-10.

Since in the game of MARS the player explores an entire planetary globe, the loop around method would be the most appropriate. This is the method used in the program.

NAMING OBJECTS

You could simply wander about, picking up OBJECT #1, and OBJECT #17, etc. Because much of the charm of adventure games comes from the

imaginative details, name all objects within the game environment, being as creative as you can.

An occasional bar of gold, or diamond is OK, but it's more fun to throw in some novelty. In the case of MARS where the player chooses between treasures and junk, some of the choices should be a little tricky and not too obviously valuable.

Table 2.1 suggests names for treasures and junk. Notice that each category has 12 items. These are the object names that will be used in the program throughout this book. If you like, you may change any or all of these.

Creating names for the supplies is a bit more straightforward. An explorer needs certain items like food and a laser gun. However, we can also throw in a few oddball items like the old magazines. Include a few ringers—supplies that serve no purpose in the game, except to load the unwary explorer down. Remember, he can only carry up to 15 objects at a time.

Table 2.2 gives a sample list of supplies. Decide what each of these items is good for, if anything, as you write the program.

Table 2.1 Suggested Names for Treasures and Junk Objects in the Game of "Mars".

#	Treasure	Junk
1	Copper Bowl	Old Shoe
2	Gold Coins	Gaudily Ornate Ring
3	Fossilized Slide Rule	Rock
4	Statue of a Martian God	Fossilized Undershorts
5	Silver Cup	Clot of Dirt
6	Glass Orb	Old Bone
7	Scroll	Sharpened Stick
8	Glimmering Stones	Chipped Urn
9	Humming Box	Petrified Wad of Bubble Gum
10	Large Sword	Colorful Flower
11	Bleached Skull	Dead Butterfly
12	Blueprint for an Ancient Martian Palace	Indescribable Slimy Thing (?)

NAMING THE MONSTERS

Dreaming up monsters to plague the player can be one of the most fun parts of creating an adventure game. Let your imagination run wild.

For now, simply name the foes. You can work out their individual characteristics in a later stage of the program development.

Table 2.2 Suggested Supply Objects for "Mars".

#	Item
1	Food
2	Bottle of Water
3	Knife
4	Gun
5	Laser
6	Coil of Rope
7	Inflatable Raft
8	Flashlight
9	Metal Pipe
10	Old Magazines
11	Compass
12	Spacesuit

Keep in mind that your monsters should be a varied lot. Too much of the same kind of battle, even if the opponents have different clever names, can very quickly become tedious.

Some monsters could be more powerful than others. Vary the ways they can be killed. Make a few beasts immune to, or even strengthened by, laser gun blasts. Some of the monsters could be beneficial if the player figures out how to take advantage of the situation and/or is lucky enough.

Table 2.3 lists the monster names used in writing the sample game of MARS.

Table 2.3 Suggested Monsters for "Mars".

Brinchley Beast
Ghost of an Ancient Martian
Grimph
Kufu
Squeanly Serpent
Purofolee

NATURAL OBSTACLES

While the heart of an adventure game is usually battling assorted monsters and villains, inanimate obstacles can add to the fun and the challenge as in MARS where the player explores an alien planet. Problematic landscapes are a natural feature of this game.

Set up a few mountains and rivers. Yes, even though Mars is an arid world, this is a fantasy game, so fudge a little. Throw in a few hidden ravines for the player to fall into. The fall could weaken the character and/or cause him to drop some of the objects he is carrying. You can rehide these dropped objects somewhere in the general vicinity.

Storms and marsquakes will complete the game.

SUMMARY

To create an adventure game, start by writing the story of the game. Where will it take place? Who is the hero/player? What is his goal? Are there any other characters?

In the initial stages of creating a plot, start thinking about some of the problems and obstacles that will make the player's task harder and, therefore, more intersting and fun.

For the sample game, MARS, the following features were considered:

- Game location—the planet Mars
- Hero/player character—an explorer from Earth
- His mission—to locate various ancient Martian treasures, and bring them aboard his spaceship to return to Earth
- Additional characters—various monsters

Now that we have a fair idea of where the game will be going, we can start writing the actual program.

Of course, you are always free to change anything in your plot as we work up the program. Sometimes you will find it difficult to implement a specific idea. Sometimes you'll come up with better ideas in mid-program. Stay flexible, although in this book the plot holds as outlined.

Even though you may change your plot at a later stage, a preliminary plot is essential. Without at least a rough blueprint, you're likely to end up with a rambling, incoherent and pointless game that's not much fun.

Chapter 3

Beginning the Program

This chapter begins to write the program for the game MARS. Monsters and the complications will come in the next chapter. (See Figure 3.1 for programming flow chart.)

The best approach to writing a complex adventure game program is to break everything down into steps, or programming modules; then concentrate on one module at a time. This minimizes the chance of getting lost in a maze of program statements and makes the task seem less intimidating.

INITIALIZATION

First identify the program and the programmer. Do this with a simple **REM** (remark) statement, like this:

```
1 REM * MARS--DELTON T. HORN *
```

The first segment of a game program usually initializes the program variables. It's a good idea to identify program modules with **REM** statements. For the first module, use a separate **REM** statement, or add the information to the program identification remark:

```
1 REM * MARS--DELTON T. HORN * INITIALIZE *
```

Figure 3.2 shows a flow chart for the initialization module of the MARS program. Initialization procedures are usually quite straight forward, but there are a few things to note.

Many computers (including the TRS-80) let you define how much memory space to reserve for string variables. In some programs, you will have to reserve more space to prevent "out of memory" or "out of string space" errors that could cause the program to bomb. On the other hand, some programs won't require much string space, so more memory is open for actual programming.

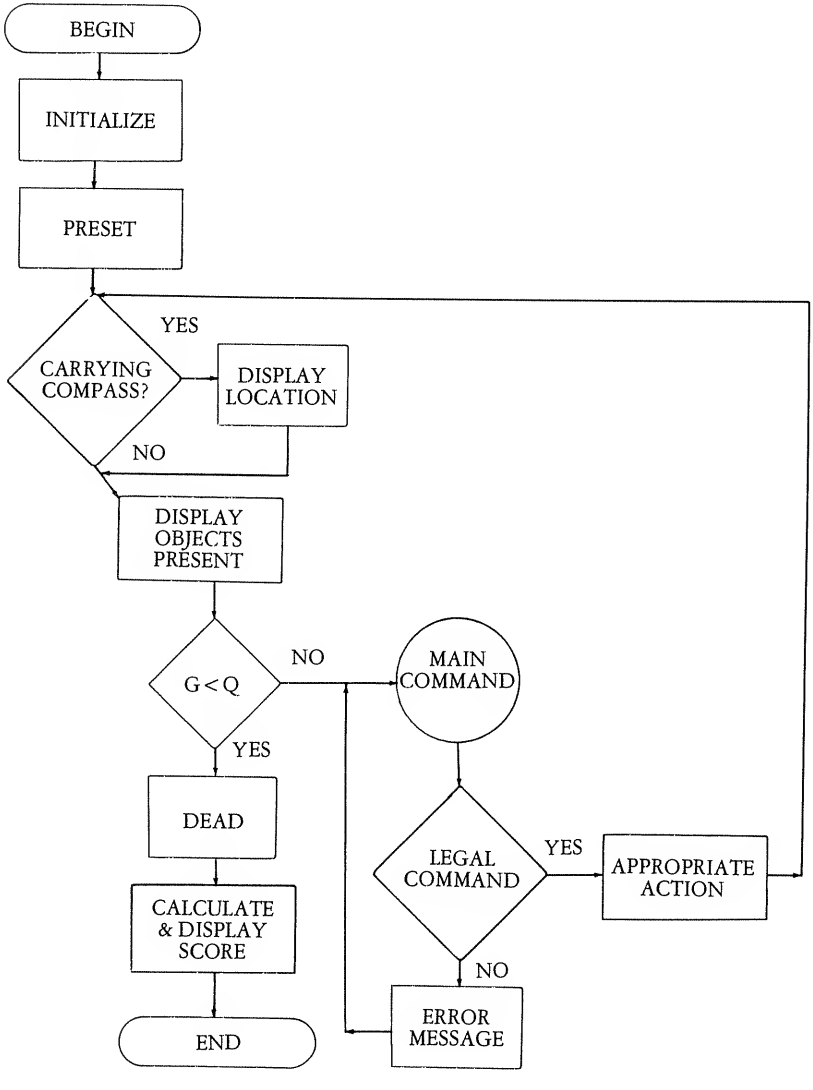


Figure 3.1 Flow-chart of the Chapter 3 Programming.

Obviously, to know how much string space you'll need to reserve, you'll have to estimate what string variables will be used in the program. In MARS you need strings to hold the player's name, and the selected commands. You'll probably also want a few additional substrings for convenience in command identification and sorting. At a guess, you should not need more than 100 characters. Since most computer systems

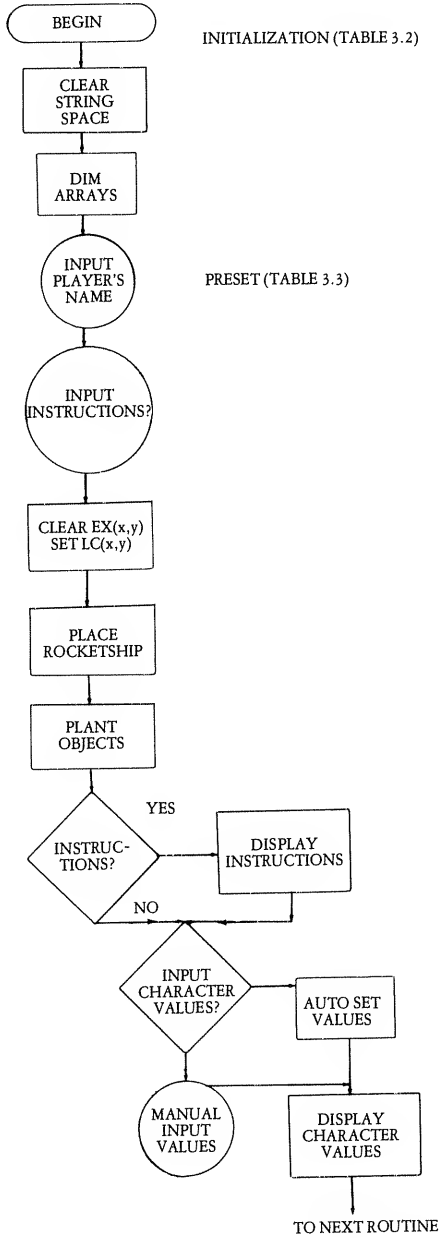


Figure 3.2 Flow-chart for INITIALIZATION and VARIABLE PRESET Routines.

(including the TRS-80) automatically clear more than that, you don't necessarily need a CLEAR statement. However, MARS will be a fairly long program, so don't waste any memory space by reserving it for unused strings. Therefore, begin the program with this statement:

```
5 CLEAR 100
```

Remember, a CLEAR command should be the first active command (other then REMarks) in any program. The CLEAR statement cancels out any previous commands.

You might want to clear different values other than 100 which is sufficient. If you clear too small an amount of string space, you may get an "out of string space" error message, and the program could bomb. If you substitute a smaller value in the CLEAR statement, bear in mind that someone else might enter a name that's longer than yours. also, inexperienced players could easily type in needlessly long commands, and it would be annoying to bomb out in mid-game because the player accidentally typed in too many characters. Don't waste memory space, but leave some elbow room.

Re-use string variables throughout the program to minimize the amount of string space used.

Next, dimension all of the arrays used in the program.

How many arrays do you need? You may not be sure at this stage, but that really isn't too much of a problem. You can always add more DIM statements as you realize the need for them. (But place all DIM statements near the beginning of the program—after the CLEAR statement, if used.)

Similarly, you may occasionally find that you've dimensioned an array that you end up not using. In this case, simply erase the unneeded DIM statement and recover the memory space reserved for that array.

In the game of MARS, I have already worked out the necessary arrays, summarized in Table 3.1.

The initialization segment of the MARS program is listed in Table 3.2.

Table 3.1 Arrays Used in the "Mars" Program.

array	size	function
A	12	supply horizontal location
B	12	supply vertical location
C	12	junk horizontal location
D	12	junk vertical location
E	12	treasure horizontal location
F	12	treasure vertical location
J	12	junk carried

S	12	supplies carried
T	12	treasures carried
LC	10,10	complete map
EX	10,10	explored areas map

Table 3.2 Initilization Portion of the "Mars" Program.

```
 1 REM * MARS * Delton T. Horn *
 5 CLEAR 100:DIM A(12): DIM B(12): DIM C(12): DIM E(12): DIM F(12)
10 DIM S(12): DIM J(12): DIM T(12): DIM LC(10,10): DIM EX(10,10)
```

INTRODUCTION AND SETTING VALUES

It is convenient, though not essential, for the computer to display some kind of introduction to the program. In the case of a farily complex game program like MARS, it will take some time for all of the variables to be preset. This makes an introductory display even more desirable. If the screen simply remains blank until all of the variables are preset, the player would wonder if the program is running, or if the computer has locked up in an error condition and could not tell without BREAKing the program, or just waiting. An introductory display, printed out correctly, however, it would indicate that the program is indeed running properly.

The variable preset routine for the game of "MARS" is listed in Table 3.3. As the flowchart in Figure 3.2 demonstrates, this section of the program is fairly straightforward.

First clear the screen, print out the name of the game, and ask for the player's name. Extra blank lines printed out make the display neater and easier to read. The player is also asked if he will want instructions for the game. This is all done in lines 20 and 30:

```
20 CLS: PRINT: PRINT" ","MARS": PRINT: INPUT "YOUR
   NAME";N$
30 PRINT: INPUT "WILL YOU NEED INSTRUCTIONS";Q$:
   Q$ = LEFT$(Q$,1)
```

The player's name is assigned to the variable N$, which will be used throughout the game.

The string variable Q$ takes on different values throughout the program. It is used to record most of the player's commands. By re-using the same variable for a number of temporary values, you can save a considerable amount of memory space.

Q$ is reduced to just its first letter with the command Q$ = LEFT$ (Q$,1). This variable will now contain a value of "Y" if the player responds

"YES" (or "YEAH", or "YEP", etc.). The computer will check and act upon the contents of this variable in a few lines.

Table 3.3 Variable Preset Routine for the "Mars" Program.

```
 20  CLS: PRINT: PRINT"  ","MARS": PRINT: INPUT "YOUR NAME";N$
 30  PRINT: INPUT "WILL YOU NEED INSTRUCTIONS";Q$:Q$ =
     LEFT$(Q$,1)
 40  FOR X = 1 TO 10: FOR Y = 1TO 10: EX(X,Y) = 0
 50  PRINT " * ";:Z = RND(17): IF Z > 12 THEN Z = 0
 60  LC(X,Y) = Z: NEXT: NEXT
 70  PRINT"  PLEASE BE PATIENT,  ";N$
 80  R1 = RND(10):R2 = RND(10): L1 = R1: L2 = R2
 90  EX(R1,R2) = 20: LC(R1,R2) = 20: PRINT: PRINT "I'M BUILDING AN
     ENTIRE PLANET HERE!": PRINT
100  FOR X = 1 TO 12: A(X) = R1: B(X) = R2
110  Y = RND(10): Z = RND(10): IF Y = R1 AND Z = R2 GOTO 110
120  C(X) = Y: D(X) = Z
130  Y = RND(10): Z = RND(10): IF Y = R1 AND Z = R2 GOTO 130
140  E(X) = Y: F(X) = Z
150  S(X) = 0: J(X) = 0: T(X) = 0
160  NEXT: GOSUB 10000: INPUT "Please press 'ENTER'  ";Q$
170  IF Q$ = "Y" GOSUB 10010
180  INPUT "ENTER 1 FOR AUTOMATIC CHARACTER OR 2 TO CREATE
     YOUR OWN";X
190  IF X = 1 GOTO 210
200  IF X = 2 GOTO 230 ELSE GOTO 180
210  AX = RND(50)+50: DX = RND(100)+100: SX = RND(50)+50:
     PX = RND(50)+50
220  GOTO 300
230  INPUT "HEALTH";DX: IF DX < 1 OR DX > 100 GOTO 230
240  DX = DX*2: INPUT "SPEED";SX: IF SX < 1 OR SX > 100 GOTO 240
250  INPUT "POWER";PX: IF PX < 1 OR PX > 100 GOTO 250
260  INPUT "AIM";AX: IF AX < 1 OR AX > 100 GOTO 260
300  CLS: PRINT: PRINT"  ",N$: PRINT
310  PRINT "HEALTH",DX/2;"%"
320  PRINT "SPEED",SX;"%"
330  PRINT "POWER",PX;"%"
340  PRINT "AIM",AX;"%"
370  DG = DX
380  INPUT "Please press 'ENTER' to play  ";Q$:CLS:PRINT: PRINT

9999  STOP
10000  FOR TT = 1 TO 321: NEXT: RETURN
10010  PRINT "INSTRUCTIONS NOT READY": RETURN
```

The next step is to clear the explored map and plant the various random obstacle marker values throughout the main location map. The explored map is represented by the array EX(10,10), and the main location map is stored in array LC(10,10). Since these are two-dimensional arrays of equal size, combine the two operations in a single pair of nested loops (X and Y). For each step through the loops, set the value of EX(X,Y) is set to zero. A random number (Z) is selected. This number may have a value of 1 to 17, but if the value is greater than 12, it is set back to 0 to represent a clear (no obstacle) space in the map. A value of 0 is five times more likely than any other specific value, but the odds are 12 to 5 that any given map location will contain some obstacle. A series of asterisks is printed to reassure the player that the program has not gotten latched up. You may omit the **PRINT** statement in line 50, but do not omit the rest of the line.

All of this is programmed in three lines, numbered 40 through 60:

```
40 FOR X = 1 TO 10:FOR Y = 1 TO 10: EX(X,Y) = 0
50 PRINT" * ";: Z = RND(17): IF Z<12 THEN Z = 0
60 LC(X,Y) = Z: NEXT: NEXT
```

The explorer's rocket ship serves as home base for this game. Remember, the object is to return the treasures to the ship and blast off to Earth. Determine a location for the rocket ship. It could always be at a fixed point, such as location 1,1, but it's more interesting to have a different randomly selected landing point for each game you play.

Since you are dealing with a two dimension map grid, identify the rocket ship's location with two simple variables, called R1 and R2. Similarly, the player's current location will be stored as L1 and L2. Since the explorer naturally starts out aboard his rocket ship, begin the game with L1 = R1 and L2 = R2.

The rocket ship's location should be marked in the map arrays. After all, we don't want to have a grimph attack on board the ship. We can do this by inserting the dummy obstacle value 20 into the appropriate map location. Any earlier value will be replaced by this value:

```
70 PRINT" PLEASE BE PATIENT, "; N$
80 R1 = RND(10): R2 = RND(10): L1 = R1: L2 = R2
90 EX(R1,R2) = 20:LC(R1,R2) = 20:PRINT: PRINT "I'M BUILDING
   AN ENTIRE PLANET HERE!": PRINT
```

Moving on down the flow chart, the next step is to plant the supply, junk, and treasure items. Since there are 12 of each stored in arrays, use a **FOR**. . .**NEXT**. . . loop to step through each one.

The supply object locations are stored in arrays A(x) and B(x). Since the supplies should naturally start out aboard the rocket ship, simply insert the values of R1 and R2 into each space in these arrays:

```
100 FOR X = 1 TO 12: A(X) = R1: B(X) = R2
```

The junk item locations are stored in arrays C(x) and D(x). Scatter these objects randomly throughout the map area, but should not appear aboard the ship. An **IF . . . THEN . . .** check is used to force the computer to select new map coordinates if it happens to duplicate the rocket ship's location:

```
110 Y = RND(10):Z = RND(10): IF Y = R1 AND Z = R2 GOTO 110
120 C(X) = Y: D(X) = Z
```

Plant the treasure objects in the same way, except use arrays E(x) and F(x):

```
130 Y = RND(10): Z = RND(10): IF Y = R1 AND Z = R2 GOTO 130
140 E(X) = Y: F(X) = Z
```

Notice that there is no provision to prevent multiple junk and/or treasure items from appearing in a single location. Statistically, this won't happen very often.

Three additional 12 space arrays (S(x), J(x), and T(x)) are used to keep track of the items the explorer character is carrying. Since at the start of the game he shouldn't be carrying anything, we'll place zeros into each of these array locations, and close the loop with the NEXT statement:

```
150 S(X) = 0: J(X) = 0: T(X) = 0
160 NEXT: GOSUB 10000: INPUT "Please press 'ENTER' ";Q$
```

Line 160 also calls a subroutine. This is a simple timing delay loop:

```
10000 FOR TT = 1 TO 321: NEXT: RETURN
```

The time delay subroutine included here gives the player a better chance to appreciate the humorous message in lines 70 and 90. Eliminate the GOSUB command from this line if you prefer, but be sure to type in the subroutine line itself. The time delay subroutine will be used throughout the program.

The structure of a computer program is usually clearer if the subroutines are kept separate from the main program. Usually, the subroutines are placed after the rest of the program. By placing the first subroutine at line 10000, you can be reasonably sure you'll have enough lines open for the main body of the program.

The line number 10000 is somewhat arbitrarily choosen, but is a good choice because it is easy to remember. Since in most game programs, a time

delay subroutine will probably be the most frequently called, place it first at the easy to remember address.

You should add a **STOP** statement just before the subroutines begin to prevent the program from accidentally crashing through to a **RETURN** without a **GOSUB** command. This can occur through an error in the programming or in sample test runs of the incomplete program. Ideally you should be able to remove this line from the finished program, but it usually stays. By giving the STOP command an odd number (9999 is used here), the beginning of the subroutines is easier to find in the line listing.

Returning to line 160, there is also an INPUT command that requests the player to press 'ENTER' without entering any data. This handy little trick lets the player manually determine how long the current information will be displayed on the screen.

You need a string variable to accept the null input. On the TRS-80 a null input (pressing 'ENTER' without typing in anything) leaves the variable with its previous value without any change. The Q$ should still hold the answer to the instructions question of line 30. If you feel uncomfortable with this, use another variable name in line 160.

By this time you have preset most of the game variables. We now check Q$ and display or skip the instructions, as appropriate:

```
170 IF Q$ = "Y" GOSUB 10010
```

Notice that this subroutine is placed immediately after the time delay subroutine already entered.

Since you often don't know all the details of a game this early in the programming, it is a good idea to leave actual instruction writing until later. The instructions for MARS are handled in a later chapter.

For now, just include a dummy subroutine to prevent bombing out during program test runs:

```
10010 PRINT "INSTRUCTIONS NOT READY: RETURN
```

To make the game more interesting, the character's success at various tasks can be dependent on certain characteristics changed from game to game. For MARS, assign ratings up to 100 for Speed, Aim, and Power (or Strength), with the variables SX, PX, and AX.

Episodes throughout the game will affect the character's health, either decreasing it (i.e., being attacked by a monster), or increasing it (i.e., eating food). You need two health variables, one to indicate the character's maximum health rating (up to 200—also his initial rating) (DX), and his constantly changing health rating (DG).

The computer can either select random values for each of the ratings or allow the player to set up his own character. A beginning player can make

things a little easier by entering 100, the maximum acceptable input, for each of the characteristics (DX is multiplied by two). Programming for both of these approaches is done in lines 180 through 260 in Table 3.3.

Display the four characteristics in percent in lines 300 to 340.

The current health rating (DG) is set equal to the maximum health rating (DX) in line 370. Line 380 is another dummy input (press 'ENTER') command so the player may study the character ratings as long as he likes before beginning the game itself and clearing the display screen.

Begin keeping a chart of all subroutines at this point. So far we only have two:

10000 TIME DELAY
10010 INSTRUCTIONS (temporary dummy)

But the number of subroutines will quickly increase and become difficult to keep track of unless you take notes.

Make up a chart of all variables used in the program to prevent reusing a variable that shouldn't be changed. The variables used thus far in the MARS program are summarized in Table 3.4. Figure 3.2 is a flow chart for the initialization and variable preset portions of the program (Tables 3.2 and 3.3).

Table 3.4 Variables Used in the Initialization and Preset Routines of the "Mars" Program (see Tables 3.2 and 3.3).

AX	Character's Aim Rating
DG	Character's Current Health Rating
DX	Character's Maximum Health Rating
L1,L2	Current Location Coordinates
PX	Character's Power Rating
R1,R2	Rocket Ship Location Coordinates
SX	Character's Speed Rating
TT	Timing Loop
X,Y,Z	misc. calculations
N$	Player's Name
Q$	Instructions? / dummy

Arrays used are outlined in Table 3.1.

BEGINNING THE MAIN PLAY ROUTINE

Now that the basic variables of the game have been preset, start programming the active part of the game—the play itself.

The main play routine is summarized in the flow chart of Figure 3.3 and is listed in Table 3.5.

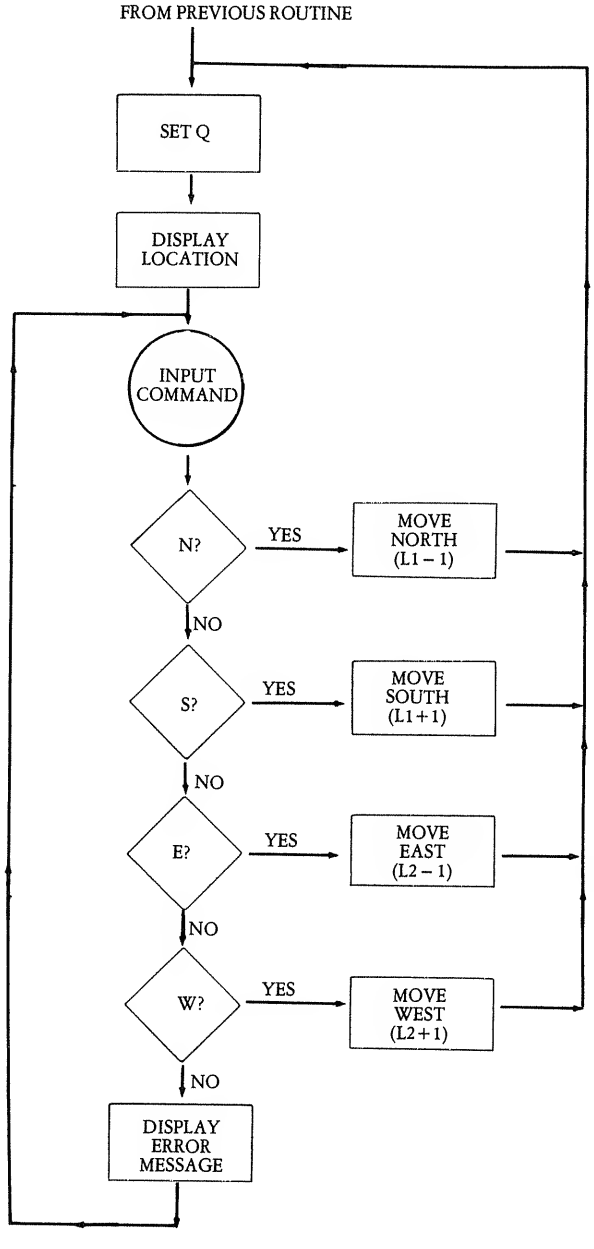


Figure 3.3 Flow-chart for MAIN PLAY routine.

As the flow chart shows, each round begins by revealing the current location, that is, telling the player where on Mars he is. This can be done quite simply as follows:

```
400 Q=LC(L1,L2): PRINT: PRINT" ","Your current coordinates
    are ";
410 PRINT L1;" : ";L2
```

Line 400 also sets variable Q equal to the obstacle number of the main map array location.

Of course, this could be combined into a single line:

```
400 Q=LC(L1,L2): PRINT: PRINT" ", "Your current coordinates
    are ";L1;" : ";L2
```

However, you may recall from Chapter 2 that one of the supply objects is a compass. In a more advanced stage of the program, we will alter this routine slightly so that the coordinates are displayed only when the character carries the compass.

Once the location coordinates are displayed, ask the player for his choice of action. Remember that later on you will add a number of monsters and obstacles that have a direct effect on the player's decisions. So leave plenty of space to add this necessary programming later. Jump down to line 700 to ask the player for his move:

```
700 Q$=" ": DG=DG-1: PRINT: PRINT "YOUR
    COMMAND, ";N$
710 INPUT Q$
```

The string variable Q$ stores the player's move. It is set to a null string (Q$ = " ") before the command request to erase any previous commands in case the player accidentally hits the **'ENTER'** key before typing in a move.

We also subtract 1 from the player's current health rating (DG). This limits how long the explorer can roam aimlessly about.

For the time being, set up the program so that it will only accept directional commands. That is, the player can only move north, south, east, or west through the map area. For simplicity, diagonal moves; i.e., southeast, are not allowed.

These commands will be in the convenient form of the initial letter only. To move north, for example, simply enter "N". The directional commands can be recognized with a series of **IF . . . THEN . . .** statements:

```
720 IF Q$="N" GOTO 5100
730 IF Q$="S" GOTO 5120
740 IF Q$="E" GOTO 5140
750 IF Q$="W" GOTO 5160
```

The line numbers for the first command (line 5100) are arbitrarily selected. Again, this is a relatively easy to remember number and leaves plenty of space for other programming. The fact that two lines will be needed for each direction determines the other three line numbers.

Since our map area is a 10 X 10 grid, move to a new location by adding or subtracting 1 to one of the location coordinates (L1 and L2). Use L1 as the north/south counter (north is L1 − 1, and south is L1 + 1). An L2 will serve as our east/west counter (east is L2 − 1, and west is L2 + 1).

Include protection to prevent the move from going out of the boundaries defined by the map, an undimensioned array location (see Chapter 2). That is, neither counter (L1 or L2) may be less than 1, nor greater than 10. Use the loop around method, as if the explorer goes completely around the planet. For example, if L1 becomes 11 after a move south, it changes to 1.

For some of the advanced plays later in the game, keep track of the player's previous location. Do this by setting up another pair of variables; i.e., L3 and L4. Set L3 equal to L1 and L4 equal to L2 before each new move is computed. Put together, the routine for moving north should look like this:

```
5100 L3 = L1:L4 = L2: L1 = L1 − 1: IF L1<1 THEN L1 = 10
5110 GOTO 400
```

Notice that after the move, the program loops back around to the coordinate display (line 400) to begin a new round.

Programming for moving south, east, and west works in the same way, and is given in Table 3.5.

The command request routine so far will recognize inputs of "N", "S", "E", or "W". What happens if the player enters something else? The program will need to display an error message if this happens and then loop back around to request a new command.

For variety, use eight different error messages, any one of which may be randomly selected. Put this into a subroutine to get error messages at other points in the game, too.

Before defaulting to the error subroutine, leave space for additional commands as you add complexity to the game. Therefore, you might number these steps like this:

```
  970 GOSUB 11700
  980 DG = DG − 1: GOTO 700
11700 X = RND(8): IF X = 1 PRINT error message #1
11710 IF X = 2 PRINT error message #2
```

* * *

```
11770 IF X = 8 PRINT error message #8
11780 RETURN
```

Alternatively, the subroutine could be programmed as follows:

```
11700 X = RND(8): ON X GOTO 11710, 11720, 11730, 11740, 11750,
      11760, 11770, 11780
11710 PRINT error message #1: RETURN
11720 PRINT error message #2: RETURN
            *  *  *
11780 PRINT error message #8: RETURN
```

Either of these methods produces exactly the same results. Some programmers prefer the **ON X GOTO**... method as more elegant and faster running. However, in this particular case, when only a single **PRINT** command is dependent on the value of X, the difference in operating speed would be an unnoticably small fraction of a second. The multiple **IF**...**THEN**... statements of the first method are clearer and easier to type in. The long string of numbers required for the **ON X GOTO**... statement invites typing errors.

When several commands are dependent on the value of X (or any variable), the **ON X GOTO**... method is certainly the way to go. In this instance, the choice is really just a matter of personal preference.

Another place that allows personal preference is in the error messages themselves. Use the messages as given in Table 3.5, or let your imagaination run free in coming up with own. Good error messages add a lot to the program's personality. Just saying "INVALID MOVE" is rather prosaic.

Remember, however, that it is annoying to play the game if the error messages are too insulting, especially if someone other than the original programmer runs the program. Use wit, not abuse.

FIRST TEST RUN

If you have carefully entered all of the program steps discussed so far (Tables 3.2, 3.3, and 3.5) you are ready to run and test the partial program. Do this at various points throughout the writing of any complex program. It will be far easier to find any errors while the program line listing is still fairly short. When the program gets longer, assume the previously tested lines are OK and concentrate on new lines.

MARS is by no means near completion. And as yet isn't even a game. The test run, while not likely to be a lot of fun, is a necessary step in writing the program.

The program listed in the tables should run properly. However, you may have a few typing errors so test thoroughly.

Table 3.6 shows a sample test run.

Make sure that entering "N" as the command reduces the first coordinate by one and leaves the second coordinate alone. Similarly, "E" should reduce the second coordinate by one and not affect the first coordinate.

```
YOUR COMMAND, DELTON? X
What on Mars are you babbling about, DELTON?
YOUR COMMAND, DELTON? NORTH
???!???
YOUR COMMAND, DELTON? N
Your current coordinates are 6 : 2
YOUR COMMAND DELTON? _____
```

Enter "S" or "W" to increase the appropriate coordinate by one. Any other entry to the "YOUR COMMAND?" query should produce an error message and no new coordinates.

Try several incorrect entries to make sure that the error messages are being selected randomly. You might get the same message twice in a row, but this is fairly unlikely. Make enough incorrect commands to see several different error messages.

Make sure that the coordinates loop around properly whenever you pass the boundaries of the map area. That is, the coordinates should work out something like this: 8 – 9 – 10 – 1 – 2 . . .

Try to return to the space ship to make sure that its coordinates remain constant.

ADDING COMMANDS

At this stage, the game is extemely dull and pointless. The player can only wander aimlessly about. So add more commands to supplement the four directional commands.

The completed game of "MARS" will use the 20 additional commands listed in Table 3.7. If you were programming your own game, you might not be sure of all of the commands you'll want to use yet, but you should make a list of the commands you expect to use. Later you can add more and/or eliminate others.

Table 3.5 Main Play/Move/Error Message Routine for the "Mars" Program.

```
399  REM * LOCATION DISPLAY *
400  Q = LC(L1,L2): PRINT: PRINT"  ","Your current coordinates are  ";
410  PRINT L1;"  :  ";L2

699  REM * MAIN COMMAND *
700  Q$ = "  ": DG = DG – 1: PRINT: PRINT "YOUR COMMAND,  ";N$;
710  INPUT Q$
```

```
  720  IF Q$ = "N" GOTO 5100
  730  IF Q$ = "S" GOTO 5120
  740  IF Q$ = "E" GOTO 5140
  740  IF Q$ =  "W" GOTO 5160

  970  GOSUB 11700
  980  DG = DG - 1: GOTO 700

 5099  REM * NORTH *
 5100  K = 0: L3 = L1: L4 = L2: L1 = L1 - 1: IF L1 < 1 THEN L1 = 10
 5110  GOTO 400
 5119  REM * SOUTH *
 5120  K = 0: L3 = L1: L4 = L2: L1 = L1 + 1: IF L1 > 10 THEN L1 = 1
 5130  GOTO 400
 5139  REM * EAST *
 5140  K = 0: L3 = L1: L4 = L2: L2 = L2 - 1: IF L2 < 1 THEN L2 = 10
 5150  GOTO 400
 5159  REM * WEST *
 5160  K = 0: L3 = L1: L4 = L2: L2 = L2 + 1: IF L2 > 10 THEN L2 = 1
 5170  GOTO 400

11699  REM * ERROR *
11700  X = RND(8): IF X = 1 PRINT "SAY WHAT?"
11710  IF X = 2 PRINT "That does not compute."
11720  IF X = 3 PRINT "DON'T BE SILLY,  ";N$;"!"
11730  IF X = 4 PRINT "???!???"
11740  IF X = 5 PRINT "I don't understand."
11750  IF X = 6 PRINT "How would that help here?"
11760  IF X = 7 PRINT "What on Mars are you babbling about,  ";N$;"?"
11770  IF X = 8 PRINT "THAT DOESN'T MAKE ANY SENSE,  ";N$;"!"
11780 RETURN
```

Table 3.6 First Test Run of the "Mars" Program.

```
     MARS
 YOUR NAME? DELTON
 WILL YOU NEED INSTRUCTIONS? Y
 ************************************************************************
 ************************************************************************
 ************************************************************************
 ************************************************************************
 ************************************************************************
 ************************************************************************
 **********                          PLEASE BE PATIENT, DELTON
 I'M BUILDING AN ENTIRE PLANET HERE!
```

```
Please press 'ENTER'  ?_
INSTRUCTIONS NOT READY
ENTER 1 FOR AUTOMATIC CHARACTER OR 2 TO CREATE YOUR OWN? 2
HEALTH? 5000
HEALTH? -1
HEALTH? 87
SPEED? 96
POWER? 7
AIM? 55
              DELTON
HEALTH        87%
SPEED         96%
POWER         7%
AIM           55%
Please press 'ENTER' to play  ? _
Your current coordinates are 7 : 9
YOUR COMMAND, DELTON? N
Your current coordinates are 6 : 9
YOUR COMMAND, DELTON? E
Your current coordinates are 6 : 8
YOUR COMMAND, DELTON? S
Your current coordinates are 7 : 8
YOUR COMMAND, DELTON? W
Your current coordinates are 7 : 9
YOUR COMMAND, DELTON? W
Your current coordinates are 7 : 10
YOUR COMMAND, DELTON? W
Your current coordinates are 7 : 1
YOUR COMMAND, DELTON? W
Your current coordinates are 7 : 2
```

Table 3.7 Commands Used in the Game of "Mars".

• Score	• Diagnosis	• Cry
• Eat	• Drink	Fill
Inflate	• Inventory	Climb
• Look	Open	• Pray
• Get	• Drop	Kill
• Help	• Wait	Touch
Map	• Blast Off	

In discussing the programming for these commands, you will notice that the line numbers jump around a bit. This is because I did not originally write the command routines in the order in which they are discussed.

First, to save a little memory space and typing, chop off the inputted command (Q$) to its first three letters (QX$) in the command identification statements. Some commands will require an object, that is, you need two words such as GET ROCK. Use the last three letters of the original command (QY$) to identify the object of the command. Before you check for the new commands, form the required substrings:

```
760 QX$ = LEFT$(Q$,3): QY$ = RIGHT$(Q$,3)
```

The entire original command is still stored as Q$, in case more information is needed from it.

Table 3.8 lists the programming for adding three new commands—SCORE, DIAGNOSIS, and CRY. These routines are flow-charted in Figure 3.4.

First, set up the scoring procedure. Treasure objects add to the score; junk objects subtract from it. The amount added or subtracted will be greater if the objects are dropped aboard the rocket ship. To summarize the scoring, say:

Carrying a treasure object = +10
Dropping a treasure object aboard the rocket ship = +12
Carrying a junk object = − 2
Dropping a junk object aboard the rocket ship = − 3.5

Add points for monsters killed in the game. The variable SV keeps track of this. Of course SV will always be equal to 0 until we add the monsters themselves (see the next chapter), but put it into the score routine now to make sure you don't forget it.

Don't bother to keep a running score each time an object is picked up or dropped. The programming could be a little awkward and a running score isn't really necessary anyway. It's easy enough to calculate the object score and add SV whenever the score is needed.

To calculate the score, check six of the arrays: J(x) (junk carried), T(x) (treasures carried), C(x) and D(x) (junk locations), and E(x) and F(x) (treasure locations). Since all of these are 12 location arrays, you can check all within a single 12 step FOR. . .NEXT loop. The calculation will be performed in a subroutine so it can easily be accessed at various points throughout the program.

```
10400 SC = 0 FOR X = 1 TO 12: IF T(X) = 1 THEN SC = SC+10
10410 IF E(X) = R1 AND F(X) = R2 THEN SC = SC+12
10420 IF J(X) = 1 THEN SC = SC − 2
10430 IF C(X) = R1 AND D(X) = R2 THEN SC = SC − 3.5
10440 NEXT: SC = SC + SV: RETURN
```

The total score is now stored as SC.

Table 3.8 Adding the Score, Diagnosis and Cry Commands to the "Mars" Program.

```
649  REM * HEALTH CHECK *
650  IF DG<1 GOTO 5070
660  IF DG<40 PRINT "You're not looking at all well, pal."
760  QX$=LEFT$(Q$,3): QY$=RIGHT$(Q$,3)
770  IF QX$="SCO" GOSUB 10400: PRINT "YOUR SCORE SO FAR IS ";
     SC: GOTO 700
780  IF QX$="DIA" GOSUB 11150: GOTO 700
790  IF QX$="CRY" GOTO 5180

5070 GOSUB 10000: PRINT: PRINT "You are deceased.": PRINT
5080 PRINT "Your final score was ";
5090 GOSUB 10400: PRINT SC: END

5179 REM * CRY *
5180 REM * HOLD SPACE TO ADD MONSTERS *
5200 PRINT "Why? Are you upset for some reason?"
5210 GOTO 650

10399 REM * SCORE CALCULATION *
10400 SC=0: FOR X= 1 TO 12: IF T(X)=1 THEN SC=SC+ 10
10410 IF E(X)=R1 AND F(X)=R2 THEN SC=SC+12
10420 IF J(X)=1 THEN SC=SC-2
10430 IF C(X)=R1 AND D(X)=R2 THEN SC=SC-3.5
10440 NEXT: SC=SC + SV: RETURN

11149 REM * DIA *
11150 X=DX * .8: IF DG>X PRINT "You're feeling just fine & dandy!":
     RETURN
11160 IF DG>150 PRINT "You feel very good.": RETURN
11170 IF DG>120 PRINT "You feel pretty good.": RETURN
11180 IF DG>105 PRINT "You're not feeling too bad, all things considered.":
     RETURN
11190 IF DG>90 PRINT "Some Alka-Seltzer might be nice. . .": RETURN
11200 IF DG>80 PRINT "You've had better days.": RETURN
11210 IF DG>70 PRINT "You're in no shape to go dancing.": RETURN
11220 IF DG>60 PRINT "You're feeling rather poorly.": RETURN
11230 IF DG>50 PRINT "Are your insurance payments up to date?": RETURN
11240 IF DG>40 PRINT "Better rehearse your moans and groans.": RETURN
11250 IF DG>30 PRINT "You're not doing well at all.": RETURN
11260 PRINT "It's a wonder you can still stand up!": RETURN
```

Now add a line to check for the "SCORE" command as the input. If it is found jump to the subroutine, print the result, and loop back around to ask for a new command:

```
770 IF QX$="SCO" GOSUB 10400: PRINT "YOUR SCORE SO
    FAR IS ";SC: GOTO 700
```

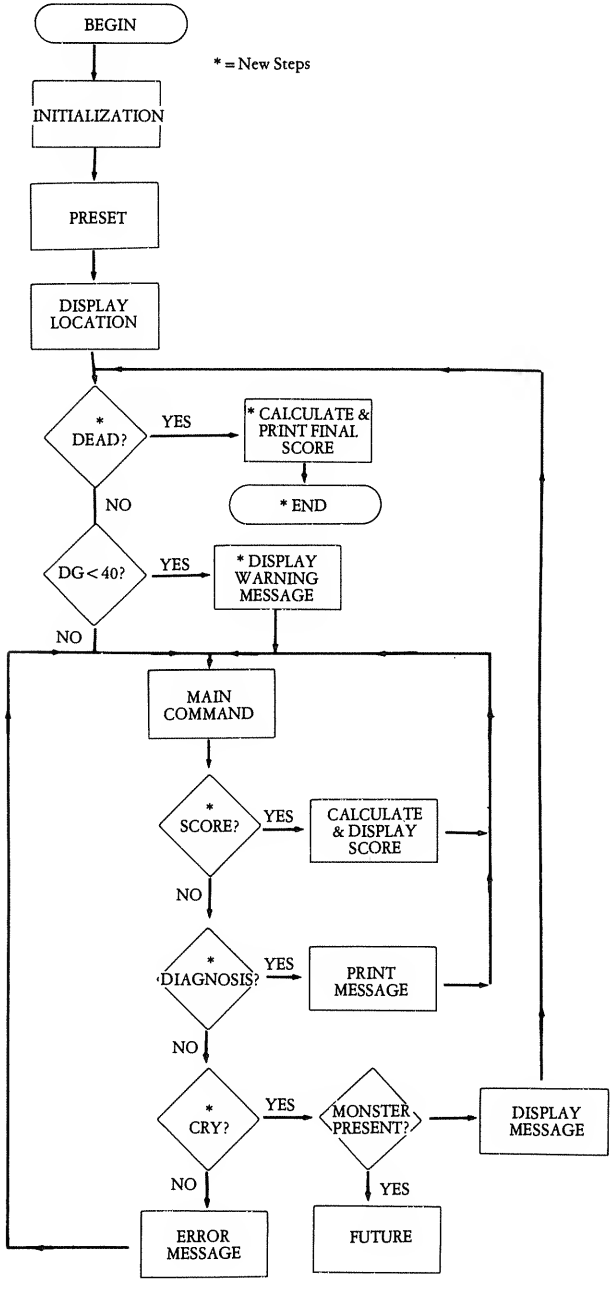


Figure 3.4 Flow-chart for SCORE/DIAGNOSIS/CRY.

The SCORE command is now programmed.

Use the variable DG to keep track of the character's current health. A number of factors throughout the game will affect this value. Obviously, if the character's health rating drops down to 0, or lower consider him dead. Check this before the command request:

```
650 IF DG<1 GOTO 5070
5070 GOSUB 10000: PRINT: PRINT "You are deceased.": PRINT
```

Subroutine 10000 is the timing delay already introduced.

To avoid frustration and to let the player know how he was doing up to the time of his demise, jump to the score calculation subroutine and print the result before ending the program:

```
5080 PRINT "Your final score was ";
5090 GOSUB 10400: PRINT SC: END
```

We should warn the player if his health rating gets too low. Line 660 prints out a warning message if the value of DG dips below 40:

```
660 IF DG<40 PRINT "You're not looking at all well, pal."
```

The DIAGNOSIS command (or DIA for short) allows the player to check the health rating whenever he wants throughout the game. This allows him to plan his strategy better. We could just print out the value of DG:

```
780 IF QX$ = "DIA" PRINT "CURRENT HEALTH RATING IS ";
    DG: GOTO 700
```

but that isn't very interesting. I prefer to use a subroutine that will print out an appropriate message depending on what range of values DG falls in:

```
780 IF QX$ = "DIA" GOSUB 11150: GOTO 700
11150 X = DX * .8: IF DG>X PRINT message #1 : RETURN
11160 IF DG>150 PRINT message #2: RETURN
11170 IF DG>120 PRINT message #3: RETURN
                  *  *  *
11250 IF DG>30 PRINT message #11: RETURN
11260 PRINT message #12: RETURN
```

The first message will be printed if the DG value is better than 80 percent of the maximum health rating (DX). The other ranges are based on absolute numbers, 150, 120, etc., but you can change these check-points if you prefer.

Placing a **RETURN** statement after each **PRINT** statement ensures that only a single message will be displayed, regardless of the value of DG.

Message 12 (line 11260) will be displayed only if DG is less than 30. An **IF...THEN...** test is not required here, because the earlier lines will have already taken care of any higher DG values.

Write your own messages, or use the ones I came up with, as they are shown in Table 3.8.

A good adventure game will result in a certain degree of frustration on the part of the player. He may want to just sit down and cry if things get too rough. Why not let the computer recognize CRY as a command, even though it is not likely to help the situation much. **CRY**ing could have special results when certain monsters are present, so we'll leave some space for the appropriate **IF...THEN...** tests. Since these would come at the beginning of the routine, put a dummy statement at the line called by the **GOTO** command:

```
790 IF QX$ = "CRY" GOTO 5180
5180 REM * HOLD SPACE TO ADD MONSTERS *
```

Ordinarily you should never have a **GOTO** or **GOSUB** command reference a line number containing only a **REM** (remark) statement. If the program gets too long, the **REM**s are the first thing to eliminate (since they don't affect the way the program runs in anyway). Going through the complete line listing and correcting **GOTO** or **GOSUB** statements is tedious at best, and you may miss one or two, causing the program to bomb.

To make sure you don't actually reference a **REM** line, use odd line numbers for all of your REM statements. For instance:

```
5179 REM * CRY *
```

The active commands are at line numbers that are multiples of 10 (for the most part), so we know just by the line number that this is a **REM** line.

However, when temporarily holding a line in an incomplete program, as we are doing here, it is OK to reference a **REM** line from a **GOTO** or **GOSUB** statement. The **REM** statement will be replaced by an active command at a later stage in the programming.

Returning to the **CRY** routine, if no monsters are present, do not let **CRY**ing have any particular effect. The computer will just display a sarcastic message, then loop back around to ask for a new command:

```
5200 PRINT "Why? Are you upset for some reason?"
5210 GOTO 650
```

Of course, you may wish to have the computer print a different message.

Tossing in a few inconsequential commands like **CRY** can increase the entertainment value of an adventure game, especially when the player doesn't know what will happen.

ADDING COMMANDS

Table 3.9 contains the programming for four more commands for the game of MARS. The new commands are **LOOK**, **PRAY**, **HELP**, and **WAIT**. As the flow-chart of Figure 3.5 shows, these commands are all quite straight-forward.

As the game grows increasingly more complex, the player may select a number of actions before moving on to a new location. The current coordinates may well be scrolled off the top of the display screen and the player could lose his location. Permit a simple command that allows the player to re-orient himself. A command of "**LOOK**" causes the program to loop around and redisplay the current coordinates:

```
860 IF QX$ = "LOO" GOTO 400
```

The **PRAY** command is similar to the **CRY** command introduced in the last section of this chapter. It will have special results when facing specific monsters, but when no monster is present, only a nonsensical message is displayed. The current health rating (DG) is also increased by 1, cancelling out the normal exertion of making a command (see line 700):

```
880 IF QX$ = "PRA" GOTO 6550
6550 REM * HOLD SPACE TO ADD MONSTERS *
6590 DG = + 1: PRINT: PRINT "GIMME THAT OLD TIME
     RELIGION!": PRINT: GOTO 650
```

Another simple command that might prove useful would be one to call up a list of all possible commands for use in the game. Some users may prefer to leave out this command and search for acceptable commands in a hit or miss fashion. Some players consider this part of the fun of learning a new adventure game. That is perfectly all right.

For those who choose to include it, the "**HELP**" command is included in Table 3.9. A subroutine is used to print out the list of possible instructions. The subroutine as given in the table includes all of the commands that will be used in the final version of the game as presented in this book. You may want to add some new ones of your own, or delete one or two. It's easy enough to add extra PRINT commands to this simple subroutine.

Note that the **HELP** subroutine includes a message stating that not all of the commands will be valid under all circumstances (line 11850). For example, you can't INFLATE your raft if you are not carrying it.

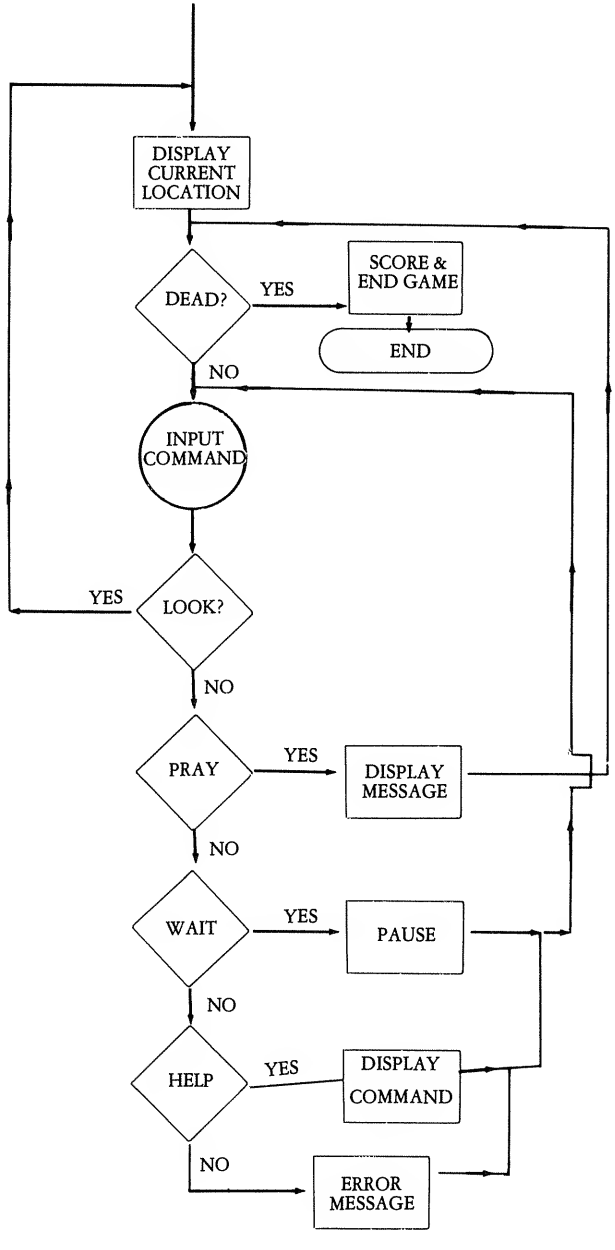


Figure 3.5 Flow-chart for LOOK/HELP/PRAY/WAIT.

Table 3.9 LOOK, HELP, PRAY, and WAIT Commands for the "Mars" Program.

```
  860  IF QX$ = "LOO" GOTO 400
  880  IF QX$ = "PRA" GOTO 6550
  920  IF QX$ = "HEL" GOSUB 11800: GOTO 700
  930  IF QX$ = "WAI" GOTO 9820

 6550  REM * HOLD SPACE FOR MONSTERS *
 6590  DG = DG + 1: PRINT: PRINT "GIMME THAT OLD TIME RELIGION!":
       PRINT: GOTO 650

 9820  PRINT "  ","(pause)": PRINT: GOSUB 10000
 9830  DG = DG + 10: IF DG > DX THEN DG = DX
 9840  GOTO 650

11799  REM * HELP *
11800  PRINT: PRINT "POSSIBLE COMMANDS INCLUDE ---"
11810  PRINT "BLAST OFF","CLIMB","CRY","DIA (diagnosis)","DRINK",
11820  PRINT "DROP", "EAT","FILL","GET","HELP","INFLATE",
11830  PRINT "INV (inventory)","KILL","LOOK","MAP","OPEN",
11840  PRINT "PRAY","SCORE","WAIT"
11850  PRINT: PRINT "Not all commands will be recognized at all times."
11860  PRINT: INPUT "Please press 'ENTER' to return to the game  ":Q$
11870  RETURN
```

Table 3.10 Routines for Displaying Object Locations in the "Mars" Program.

```
  440  Y = 0; FOR X = 1 TO 12: IF A(X) = L1 AND B(X) = L2 GOSUB 10450
  450  IF C(X) = L1 AND D(X) = L2 GOSUB 10570
  460  IF E(X) = L1 AND F(X) = L2 GOSUB 10700
  470  IF Y > 8 THEN Y = 0: INPUT "Please press 'ENTER'  ";Q$
  480 NEXT X

10449  REM * SUPPLIES PRESENT *
10450  IF X = 1 PRINT "Some food is here."
10460  IF X = 2 PRINT "A bottle of water is here."
10470  IF X = 3 PRINT "A knife is here."
10480  IF X = 4 PRINT "A gun is here."
10490  IF X = 5 PRINT "A laser is here."
10500  IF X = 6 PRINT "A coil of rope is here."
10510  IF X = 7 PRINT "An inflatable raft is here."
10520  IF X = 8 PRINT "A flashlight is here."
10530  IF X = 9 PRINT "A metal pipe is here."
10540  IF X = 10 PRINT "Some old magazines are here."
10550  IF X = 11 PRINT "A compass is here."
10560  IF X = 12 PRINT "Your spacesuit is hanging neatly on its rack."
10565  Y = Y + 1: RETURN
```

```
10569  REM * JUNK PRESENT *
10570  IF X = 1 PRINT "An old shoe is here."
10580  IF X = 2 PRINT "A gaudily ornate ring is here."
10590  IF X = 3 PRINT "A rock is here."
10600  IF X = 4 PRINT "A pair of fossilized undershorts is here."
10610  IF X = 5 PRINT "A large clot of dirt is here."
10620  IF X = 6 PRINT "An old bone is here."
10630  IF X = 7 PRINT "A sharpened stick is here."
10640  IF X = 8 PRINT "A badly chipped urn is here."
10650  IF X = 9 PRINT "A petrified wad of bubble gum is here."
10660  IF X = 10 PRINT "A colorful flower is here."
10670  IF X = 11 PRINT "A dead butterfly is here."
10680  IF X = 12 PRINT "An indescribably slimy thing is here."
10690  Y = Y + 1: RETURN
10699  REM * TREASURES PRESENT *
10700  IF X = 1 PRINT "A dented copper bowl is here."
10710  IF X = 2 PRINT "Some gold coins are here."
10720  IF X = 3 PRINT "A fossilized slide rule is here."
10730  IF X = 4 PRINT "A statue of a three-armed Martian god is here."
10740  IF X = 5 PRINT "A tarnished silver cup is here."
10750  IF X = 6 PRINT "A glass orb is here."
10760  IF X = 7 PRINT "A scroll inscribed with the ancient Martian languge is
       here."
10770  IF X = 8 PRINT "Some glittering stones are here."
10780  IF X = 9 PRINT "A mysteriously humming box is here."
10790  IF X = 10 PRINT "A large, polished sword is here."
10800  IF X = 11 PRINT "A bleached skull is here."
10810  IF X = 12 PRINT "A set of blueprints for an ancient Martian palace is here."
10820  Y = Y + 1: RETURN
```

So far we have been decrementing the health rating (DG) on each command (except **PRAY**), but we haven't allowed any way for the character to recover. One way to recover would be to stop and rest for a while. The "**WAIT**" command allows the player to choose this option.

When the "**WAIT**" command is entered, "(pause)" is displayed and the timing subroutine (10000) is called. The DG current health rating is increased by 10. An **IF**. . . **THEN** test is included in line 9830 to prevent the current health rating (DG) from exceeding the character's maximum health level (DX).

After all of this is done, the program loops back around to ask for a new command:

```
930 IF QX$ = "WAI" GOTO 9820
9820 PRINT " ","(pause)": PRINT: GOSUB 10000
9830 DG = DG + 10: IF DG > DX THEN DG = DX
9840 GOTO 650
```

FINDING THE OBJECTS

Since the object of the game of MARS is to collect ancient Martian treasures, you obviously need a routine to identify these objects when you encounter them.

Three pairs of 12 locations check against the player's current coordinates. Do this with a simple **FOR. . .NEXT. . .** loop, like this:

```
440 Y = 0: FOR X = 1 TO 12: IF A(X) = L1 AND B(X) = L2 GOSUB
    10450
450 IF C(X) = L1 AND D(X) = L2 GOSUB 10570
460 IF E(X) = L1 AND F(X) = L2 GOSUB 10700
480 NEXT X
```

The purpose of the variable Y will be explained shortly.

Each of the subroutines called from this loop will print out one of 12 messages (determined by the current value of X) to identify the object present.

Since many different objects may be at a single location, especially aboard the rocket ship in later stages of the game, some lines could be scrolled off the top of the screen before the player has a chance to read them. The variable Y is used as a counter to eliminate this problem.

Each time one of the subroutines is called, it adds 1 to Y before returning control to the main body of the program, for example:

```
10565 Y = Y+1: RETURN
```

If Y accumulates a value greater than 8, it is set back to 0 and the computer stops until the player presses the 'ENTER' key, indicating he is ready for more information:

```
470 IF Y>8 THEN Y = 0: INPUT "Please press 'ENTER' ";Q$
```

You might want to try a sample run at this point. Make sure that all 12 supply items appear at the beginning location (aboard the rocket ship), and that the treasure and junk items are scattered randomly throughout the rest of the map. Try out all of the commands added since the last sample run.

Well, now you can see the various objects, but you can't do anything with them. A new command, **"GET"**, will allow the explorer to pick up the various items. Objects can be gotten rid of by using the **"DROP"** command.

The programming for the **"GET"** and **"DROP"** commands is listed in Table 3.11, and flow-charted in Figure 3.6. While this routine is rather long, it is not complicated.

For a **GET** command, G is set to a value of 1. This variable takes on a value of 2 for a **DROP** command.

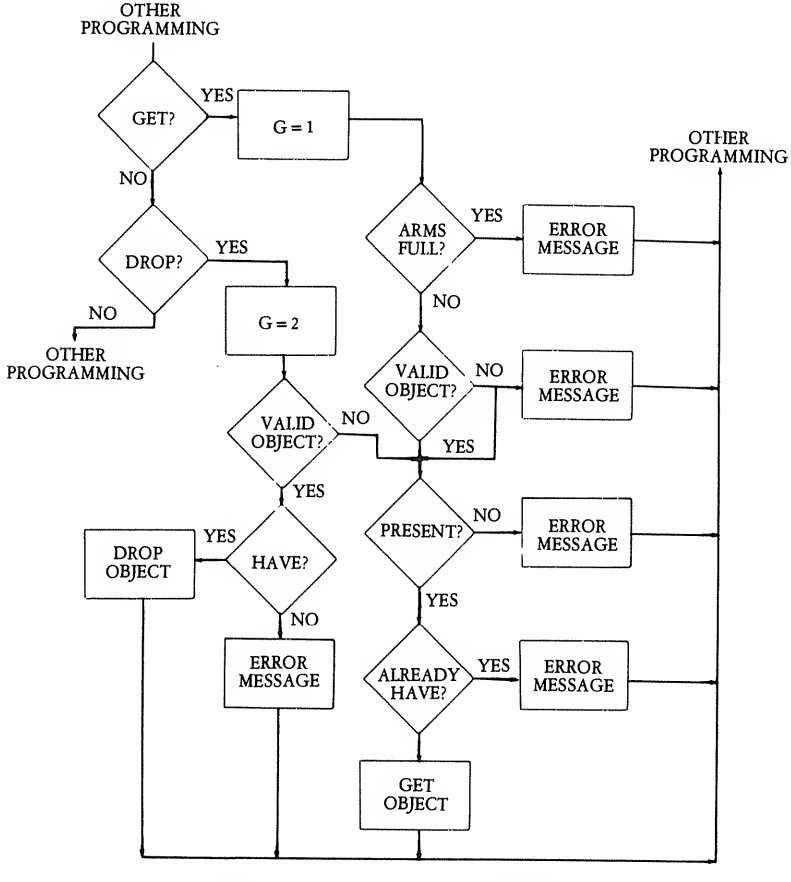


Figure 3.6 Flow-chart for GET/DROP.

Table 3.11 Adding the GET and DROP Commands to the "Mars" Program.

```
 500  REM * HOLD *
 890  IF QX$ = "GET" GOTO 6810
 900  IF QX$ = "DRO" GOTO 6910

6809  REM * GET/DROP *
6810  IF Q$ = "GET" PRINT "GET WHAT?": GOTO 650
6820  IF QY$ = "OST" GOTO 6870
6830  IF Q$ = "GET SICK" GOTO 6880
6840  IF QY$ = "ENT" OR QY$ = "KED" GOTO 6890
6850  IF QY$ = "OWN" GOTO 6900
```

```
6860  G = 1: Y = 0: FOR X = 1 TO 12: Y = Y + S(X) + J(X) + T(X): NEXT: IF
      Y > 17 GOTO 9850 ELSE GOTO 6930
6870  PRINT "I think you already are. . . ": GOTO 500
6880  PRINT "That's disgusting!": GOTO 500
6890  PRINT "SAME TO YOU,  ";N$;"!": GOTO 500
6900  PRINT "This is no time to boggie!": GOTO 500
6910  IF QY$ = "EAD" GOTO 6890
6920  G = 2
6930  IF QY$ = "OOD" U = 1: GOTO 7500
6940  IF QY$ = "TLE" OR QY$ = "TER" U = 2: GOTO 7500
6950  IF QY$ = "IFE" U = 3: GOTO 7500
6960  IF QY$ = "GUN" U = 4: GOTO 7500
6970  IF QY$ = "SER" U = 5: GOTO 7500
6980  IF QY$ = "OPE" OR QY$ = "OIL" U = 6: GOTO 7500
6990  IF QY$ = "AFT" U = 7: GOTO 7500
7000  IF QY$ = "GHT" U = 8: GOTO 7500
7010  IF QY$ = "IPE" U = 9: GOTO 7500
7020  IF RIGHT$(Q$,4) = "INES" U = 10: GOTO 7500
7030  IF QY$ = "ASS" U = 11: GOTO 7500
7040  IF QY$ = "UIT" AND G = 2 GOTO 5070
7050  IF QY$ = "UIT" U = 12: GOTO 7500
7060  IF QY$ = "HOE" U = 1: GOTO 7600
7070  IF RIGHT$(Q$,4) = "RING" U = 2: GOTO 7600
7080  IF QY$ = "OCK" U = 3: GOTO 7600
7090  IF QY$ = "RTS" U = 4: GOTO 7600
7100  IF QY$ = "IRT" OR QY$ = "LOT" U = 5: GOTO 7600
7110  IF QY$ = "ONE" U = 6: GOTO 7600
7120  IF QY$ = "ICK" U = 7: GOTO 7600
7130  IF QY$ = "URN" U = 8: GOTO 7600
7140  IF QY$ = "WAD" OR QY$ = "GUM" U = 9: GOTO 7600
7150  IF QY$ = "WER" U = 10: GOTO 7600
7160  IF QY$ = "FLY" U = 11: GOTO 7600
7170  IF RIGHT $(Q$,4) = "HING" U = 12: GOTO 7600
7180  IF QY$ = "PER" OR QY$ = "OWL" U = 1: GOTO 7650
7190  IF QY$ = "INS" U = 2: GOTO 7650
7200  IF QY$ = "ULE" U = 3: GOTO 7650
7210  IF QY$ = "TUE" OR QY$ = "GOD" U = 4: GOTO 7650
7220  IF QY$ = "VER" OR QY$ = "CUP" U = 5: GOTO 7650
7230  IF QY$ = "ORB" U = 6: GOTO 7650
7240  IF QY$ = "OLL" U = 7: GOTO 7650
7250  IF RIGHT$(Q$,4) = "ONES" U = 8: GOTO 7650
7260  IF QY$ = "BOX" U = 9: GOTO 7650
7270  IF QY$ = "ORD" U = 10: GOTO 7650
7280  IF QY$ = "ULL" U = 11: GOTO 7650
7290  IF QY$ = "NTS" U = 12: GOTO 7650
```

```
7300  X = LEN(Q$): IF X < 6 GOTO 7350
7310  IF G = 1 THEN X = X - 4 ELSE X = X - 5
7320  QY$ = RIGHT$(Q$,X)
7330  PRINT "SORRY,  ";N$:", BUT I DON'T SEE ANY  ";QY$; "HERE."
7350  PRINT "Please try to keep your commands rational in the future."
7360  PRINT: DG = DG - 3: GOTO 500
7500  IF G = 2 GOTO 7550
7510  IF S(U) > 0 GOTO 7540
7515  IF A(U) = L1 AND B(U) = L2 GOTO 7520 ELSE GOTO 7590
7520  S(U) = 1: A(U) = 0: B(U) = 0: PRINT "OK": GOTO 500
7540  PRINT "You already have it!": DG = DG - 1: GOTO 500
7550  IF S(U) < 1 GOTO 7580
7560  S(U) = 0: A(U) = L1; B(U) = L2; PRINT "OK": GOTO 500
7580  PRINT "You don't have it!": DG = DG - 1: GOTO 500
7590  PRINT "IT'S NOT HERE,  ";N$: DG = DG - 1: GOTO 500
7600  IF G = 2 GOTO 7630
7610  IF J(U) > 0 GOTO 7540
7615  IF C(U) = L1 AND D(U) = L2 GOTO 7620 ELSE GOTO 7590
7620  J(U) = 1: C(U) = 0: D(U) = 0: PRINT "OK": GOTO 500
7630  IF J(U) < 1 GOTO 7580
7640  J(U) = 0: C(U) = L1: D(U) = L2: PRINT "OK": GOTO 500
7650  IF G = 2 GOTO 7680
7660  IF T(U) > 0 GOTO 7540
7665  IF E(U) = L1 AND F(U) = L2 GOTO 7670 ELSE GOTO 7590
7670  T(U) = 1: E(U) = 0: F(U) = 0: PRINT "OK": GOTO 500
7680  IF T(U) < 1 GOTO 7580
7690  T(U) = 0: E(U) = L1: F(U) = L2; PRINT "OK": GOTO 500

9850  PRINT "Your arms are full. You can't carry anything more unless"
9860  PRINT "you drop something.": DG = DG - 1: GOTO 500
```

An object must be specified in the command. Just saying "GET" doesn't mean much. You must use two words; for instance, GET KNIFE.

Lines 6820 through 6850, and 6910 check for wise guy commands (such as "GET LOST" or "DROP DEAD") and prints out appropriate messages (lines 6870 through 6900). They are not essential to the operation of the program, and may be eliminated or changed as you like.

QY$ was set to the last three letters of the complete command (Q$) in an earlier portion of the program. This string variable is now put to use to identify the object being acted on. Lines 6930 through 7290 check for a match with each of the 36 supply, junk, and treasure objects in the game. When a match is found the variable U is set to an appropriate value, and control is switched to a routine for acting on the appropriate arrays. The supply array routine begins at line 7500, the junk arrays at 7600, and the treasure arrays at 7650.

Some items may be identified by different names. For example, a player might try to pick up the coil of rope by entering "GET COIL" or "GET ROPE". The program should recognize either name:

```
6980 IF QY$ = "OPE" OR QY$ = "OIL" U = 6: GOTO 7500
```

You must also watch out for different item names that end in the same three letters, such as OLD MAGAZINES and GLITTERING STONES. To positively identify these items, you will need to check the last four letters of the original command (Q$):

```
7020 IF RIGHT$(Q$,4) = "INES" U = 10: GOTO 7500
7250 IF RIGHT$(Q$,4) = "ONES" U = 8: GOTO 7650
```

If the player enters an item that is not included in the recognized object list, we will need to fall into an error routine. First we subtract GET or DROP from the original complete command to isolate the object name:

```
7300 X = LEN(Q$): IF X < 6 GOTO 7350
7310 IF G = 1 THEN X = X - 4 ELSE X = X - 5
7320 QY$ = RIGHT$(Q$,X)
```

For a **GET** command the first four letters are subtracted (the word **GET**, and the between word space). For a **DROP** command we delete the first five letters (D—R—O—P, and a space). You can now include the unrecognized object name in the error message:

```
7330 PRINT "SORRY, ";N$;", BUT DON'T SEE ANY "; QY$;
     " HERE."
7350 PRINT "Please try to keep your commands rational in the
     future."
7360 PRINT: DG = DG - 3: GOTO 500
```

Notice that there is also a penalty to the current health rating (DG).

If the player enters "**GET DRUNK**" as his command, the computer will reply:

```
SORRY, DELTON, BUT I DON'T SEE ANY DRUNK HERE.
Please try to keep your commands rational in the future.
```

Now, let's see what happens when a valid object name is entered with regard to the routine for the supply arrays. The junk and treasure routines work the same way.

Let's say the command is "**GET FOOD**". There are three possible ways the computer may respond depending on current conditions:

* FOOD is present

* FOOD is not present
* Character is already carrying FOOD

The S(X) array indicates which supply objects the player is already carrying; so check the appropriate supply location to determine if this is the case:

```
7510 IF S(U)>0 GOTO 7540
7540 PRINT "You already have it!": DG=DG - 1: GOTO 500
```

If the character is not carrying the FOOD, check the location arrays to see if the FOOD and the player are in the same area:

```
7515 IF A(U)=L1 AND B(U)=L2 GOTO 7520 ELSE GOTO 7590
```

If the specified object is not present, we once again have an error condition:

```
7590 PRINT "IT'S NOT HERE, ";N$: DG=DG - 1: GOTO 500
```

Assuming the FOOD is present, clear the appropriate location arrays and mark the appropriate point in the supplies array:

```
7520 S(U)=1: A(U)=0: B(U)=0: PRINT "OK": GOTO 500
```

The displayed message "OK" confirms that the command has been obeyed.

If the command is DROP FOOD, there are only two possibilities. The character carrying the FOOD can drop it; otherwise, an error message is displayed. **DROP**ping an object is just like **GET**ting, except in reverse. The player's current coordinates are placed into the appropriate object location arrays, and the object is deleted from the supplies carried array:

```
7500 IF G=2 GOTO 7550
7550 IF S(U)<1 GOTO 7580
7560 S(U)=0: A(U)=L1: B(U)=L2: PRINT "OK": GOTO 500
7580 PRINT "You don't have it!": DG = DG - 1: GOTO 500
```

To make the game a little harder, allow the explorer to carry only up to 17 items at a time. This routine will only be appropriate for GET commands. The player can **DROP** as many items as he is carrying:

```
6860 G=1: Y=0: FOR X= 1 TO 12: Y= Y + S(X) + J(X) + T(X):
     NEXT: IF Y>17 GOTO 9850 ELSE GOTO 6930
9850 PRINT "Your arms are full. You can't carry anything more
     unless"
9860 PRINT "you drop something.": DG=DG-1: GOTO 500
```

At the beginning of each game, the explorer starts out aboard the rocket ship with all 12 supplies present. He will probably want to GET most of them. Entering 12 separate commands is a bit of a nuisance, so allow a **GET ALL** command to gather the supplies aboard the rocket ship but under no other conditions. The programming needed to accomplish this is listed in Table 3.12.

Two of the supplies are of special significance. As stated earlier, the current coordinates should be displayed only when the player is carrying the COMPASS. You can easily change line 410 to include an **IF...THEN...** test:

```
410 IF S(11) = 0 PRINT "? : ?" ELSE PRINT L1;" : ";L2
```

Table 3.12 The "GET ALL" Routine for the "Mars" Program.

```
6815 IF Q$ = "GET ALL" GOTO 9900
9899 REM * GET ALL *
9900 IF L1 = R1 AND L2 = R2 GOTO 9920
9910 GOTO 970
9920 FOR X = 1 TO 12: IF A(X) = L1 AND B(X) = L2 GOTO 9950
9930 NEXT: PRINT "SUPPLIES GATHERED." : GOTO 500
9950 A(X) = 0: B(X) = 0: S(X) = 1: GOTO 9930
```

Table 3.13 Special Programming for the Compass and the Spacesuit in the Game of "Mars".

```
410  IF S(11) = 0 PRINT "? : ?" ELSE PRINT L1;" : ";L2
420  IF L1 = R1 AND L2 = R2 GOTO 430
425  IF S(12) = 0 GOTO 5000
430  IF L1 = R1 AND L2 = R2 PRINT "You are safely aboard your rocket ship.":
     DG = DG + 3
5000 PRINT "You are outside without your spacesuit!": PRINT
5010 FOR X = 1 TO 4: Y = RND(5): Z = RND(75) + 25: FOR ZZ = 1 TO Z: NEXT
     ZZ
5020 IF Y = 1 PRINT "* gasp *",
5030 IF Y = 2 PRINT "* choke *",
5040 IF Y = 3 PRINT "* pant-pant *",
5050 IF Y = 4 PRINT "* wheeze *",
5060 NEXT X: PRINT

7040 IF QY$ = "UIT" AND G = 2 GOTO 5000
```

It is logical to assume that the explorer can only survive outside his rocket ship if he is wearing his space suit. Leaving the ship without the suit, or dropping the suit, becomes fatal by adding the steps shown in Table 3.13.

Also notice that being aboard the rocket ship is beneficial, in that 3 is added to the current health rating (DG) of the character.

INVENTORY

Since it is easy to lose track as the game goes on of what your explorer is carrying, add a command to display a list of the items being carried. Call this command "INVENTORY", or "INV" for short.

To program in the INVENTORY function to the game of MARS see the list in Table 3.13. In this simple routine each object carried array location is checked. If the value is 1, the name of the appropriate object is displayed.

Various objects are listed in a random order to help block attempts to distinguish between junk and treasure items based on their displayed position.

Table 3.14 Adding the INVENTORY Command to the "Mars" Program.

```
  840  IF QX$ = "INV" GOSUB 11300: GOTO 700
11299  REM * INVENTORY *
11300  PRINT: PRINT "YOU ARE NOW CARRYING ---"
11310  IF S(1) = 1 PRINT "FOOD ",
11320  IF J(2) = 1 PRINT "ORNATE RING ",
11330  IF T(3) = 1 PRINT "FOSSILIZED SLIDE RULE ",
11340  IF J(4) = 1 PRINT "FOSSILIZED UNDERSHORTS ",
11350  IF S(5) = 1 PRINT "LASER ",
11360  IF J(6) = 1 PRINT "OLD BONE ",
11370  IF T(7) = 1 PRINT "SCROLL ",
11380  IF J(8) = 1 PRINT "URN ",
11390  IF S(9) = 1 PRINT "METAL PIPE ",
11400  IF J(10) = 1 PRINT "FLOWER ",
11410  IF T(11) = 1 PRINT "SKULL ",
11420  IF J(12) = 1 PRINT "SLIMY THING (?) ",
11430  IF S(11) = 1 PRINT "COMPASS ",
11440  IF T(10) = 1 PRINT "LARGE SWORD ",
11450  IF J(9) = 1 PRINT "PETRIFIED WAD OF BUBBLE GUM ",
11460  IF T(8) = 1 PRINT "GLITTERING STONES ",
11470  IF S(7) = 1 PRINT "INFLATABLE RAFT ",
11480  IF T(6) = 1 PRINT "GLASS ORB ",
11490  IF J(5) = 1 PRINT "CLOT OF DIRT ",
11500  IF T(4) = 1 PRINT "STATUE OF MARTIAN GOD ",
```

```
11510  IF S(3) = 1 PRINT "KNIFE ",
11520  IF T(2) = 1 PRINT "GOLD COINS ",
11530  IF J(1) = 1 PRINT "OLD SHOE ",
11540  IF T(1) = 1 PRINT "COPPER BOWL ",
11550  IF S(2) = 1 PRINT "BOTTLE OF WATER ",
11570  IF J(3) = 1 PRINT "ROCK ",
11580  IF S(4) = 1 PRINT "GUN ",
11590  IF T(5) = 1 PRINT "SILVER CUP ",
11600  IF S(6) = 1 PRINT "COIL OF ROPE ",
11610  IF J(7) = 1 PRINT "SHARPENED STICK ",
11620  IF S(8) = 1 PRINT "FLASHLIGHT ",
11630  IF T(9) = 1 PRINT "MYSTERIOUSLY HUMMING BOX ",
11640  IF S(10) = 1 PRINT "OLD MAGAZINES ",
11650  IF J(11) = 1 PRINT "BUTTERFLY ",
11660  IF T(12) = 1 PRINT "BLUEPRINTS ",
11670  IF S(12) = 1 PRINT "SPACESUIT ",
11680  Z = 0: FOR Y = 1 TO 12: Z = Z + S(Y) + J(Y) + T(Y): NEXT: IF Z = 0
       PRINT "nothing",
11690  PRINT: RETURN
```

BLAST OFF

As you know the way to win the game is to bring as many Martian treasures as possible aboard the rocket ship and blast off for Earth. Obviously, you need a blast off command.

The game of MARS ends when either the explorer gets killed, or the player uses the BLAST OFF command.

The player should only be able to **BLAST OFF** aboard the rocket ship. If this command is entered at any other location, a sarcastic message is displayed (line 9970).

Table 3.15 Adding the BLAST OFF Command to the Game of "Mars".

```
 960  IF QX$ = "BLA" GOTO 9960

9960  IF L1 = R1 AND L2 = R2 GOTO 9980
9970  PRINT "You don't have jet propulsion engines in your shoes!": DG = DG -
      2: GOTO 500
9980  FOR X = 1 TO 100: PRINT " * ";: FOR Y = 1 TO 55: NEXT: NEXT
9985  PRINT: PRINT: GOSUB 10400
9990  PRINT " Your score was ";SC: IF SC > 75 PRINT "FANTASTIC
      WORK, ";N$;"!"
9995  IF SC < 25 PRINT "I'm not too impressed by your performance. . ."
9997  END
```

When the player chooses to **BLAST OFF** aboard the rocket ship, the final score will be displayed. A score of better than 75 will result in a congratulatory message from the computer. If the score is below 25, the computer will express dissatisfaction.

The **BLAST OFF** routine is listed in Table 3.15.

THE EAT AND DRINK COMMANDS

Since the supplies include FOOD and a BOTTLE OF WATER, include EAT and DRINK commands. Programming for these commands is flowcharted in Figure 3.7 and listed in Table 3.16.

EATing or DRINKing boosts the character's current health rating (DG), but will not let it surpass the maximum health rating (DX).

In the next chapter, add the monsters. Some dead monsters may be edible, so the EAT command must consist of two words to identify what is to be eaten.

To **EAT** in front of certain live monsters will have special results, so we will hold a space for the appropriate IF...THEN... tests to be added later:

```
 800 IF QX$ = "EAT" GOTO 5300
5300 REM * HOLD FOR MONSTERS *
5340 X = LEN(Q$): IF X < 5 GOTO 5590
5590 PRINT "Eat what?": DG = DG - .25: GOTO 500
```

If EAT FOOD is specified, the computer must check to see if the explorer is actually carrying any food (S(1)):

```
5310 IF QY$ = "OOD" GOTO 5420
5420 IF S(1) = 1 GOTO 5450
5430 PRINT "YOU DON'T HAVE ANY!"
5440 DG = DG - 1: GOTO 500
```

If the explorer does have the food, it will be deleted from the array since you can't eat the same food twice, and the current health rating (DG) will be increased by up to 25% of the maximum (DX), without exceeding it:

```
5450 PRINT " ", "burp": PRINT
5460 DG = DG + (DX * .25): IF DG > DX THEN DG = DX
5470 S(1) = 0: GOTO 500
```

Lines 5350 through 5410 display one of six randomly selected error messages in case the player tries to have his character EAT something the computer does not recognize as edible (for example, "EAT FOOT"). At this point only FOOD is recognized, but a few other items will be added in the next chapter.

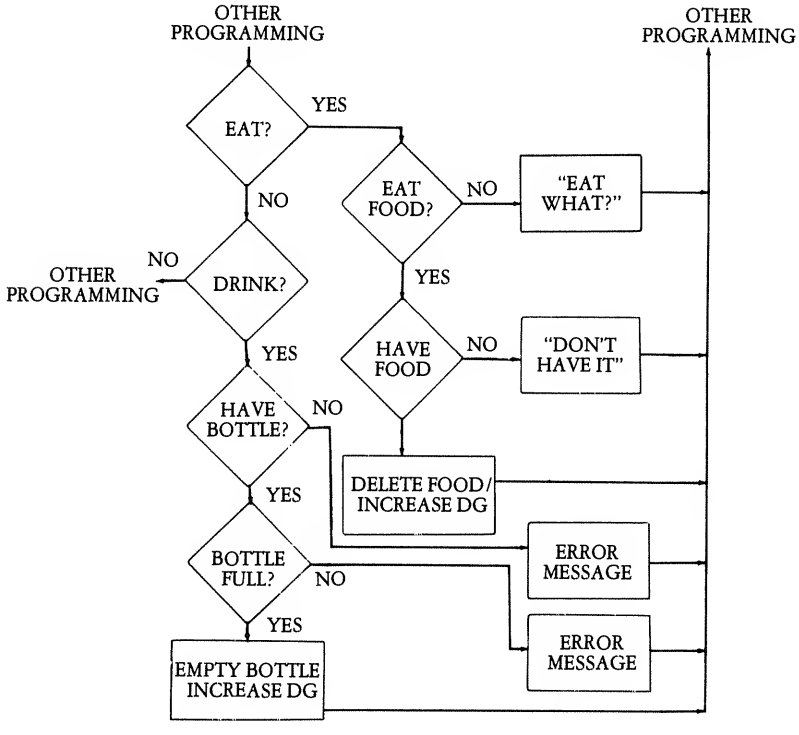


Figure 3.7 Flow-chart for EAT/DRINK.

Table 3.16 Adding the EAT and DRINK Commands to the "Mars" Program.

```
800  IF QX$ = "EAT" GOTO 5300
810  IF QX$ = "DRI" GOTO 5700

5299 REM * EAT *
5300 REM * HOLD FOR MONSTERS *
5310 IF QY$ = "OOD" GOTO 5420
5340 X = LEN(Q$): IF X < 5 GOTO 5590
5350 X = RND(6): DG = DG - .5: IF X = 1 PRINT "Eat WHAT???"
5360 IF X = 2 PRINT "—er— No, thank you..."
5370 IF X = 3 PRINT "ARE YOU NUTS?"
5380 IF X = 4 PRINT "That hungry, I'm not!"
5390 IF X = 5 PRINT "YUK!"
5400 IF X = 6 PRINT "Has anyone ever told you that you have weird tastes?"
5410 GOTO 500
```

```
5420  IF S(1) = 1 GOTO 5450
5430  PRINT "YOU DON'T HAVE ANY!"
5440  DG = DG - 1: GOTO 500
5450  PRINT "  ","burp": PRINT
5460  DG = DG+(DX * .25): IF DG > DX THEN DG = DX
5470  S(1) = 0: GOTO 500
5590  PRINT "Eat what?": DG = DG - .25: GOTO 500

5699  REM * DRINK *
5700  REM * HOLD FOR RIVER *
5710  IF S(2) = 1 GOTO 5740
5720  IF S(2) = 2 GOTO 5800
5730  PRINT "THERE IS NOTHING HERE TO DRINK, ";N$:
      DG = DG - .5: GOTO 500
5740  PRINT "  ",: FOR X = 1 TO 3: FOR Y = 1 TO 85: NEXT Y
5750  PRINT"* glug *",:NEXT X: PRINT
5760  GOSUB 10000
5770  PRINT "  ","AHHHH!":DG = DG + (DX * .15)
5780  IF DG > DX THEN DG = DX
5790  S(2) = 2; GOTO 500
5800  PRINT "Your water bottle is empty.": DG = DG - .4: GOTO 500

11560 IF S(2) = 2 PRINT "EMPTY BOTTLE ",
```

The **DRINK** command is basically similar to the EAT command. The command is only valid if the explorer has the bottle of water, or is at the RIVER—to be added in the next chapter.

After drinking, the water should be gone, but the bottle would logically still be there. By inserting a value of 2 into array location S(2), you can replace the BOTTLE OF WATER with an EMPTY BOTTLE. Be sure to add this line to the INVENTORY subroutine:

```
11560 IF S(2) = 2 PRINT "EMPTY BOTTLE"
```

DRINKing from an EMPTY BOTTLE is, of course, not permitted. In the next chapter, you will add a command to refill the bottle at the RIVER.

There is another "magical" way to refill the bottle. See if you can figure out how to refill your water bottle without a river.

Table 3.17 Routines and Subroutines for the "Mars" Program Presented in Chapter 3.

Routines	
1—10	initialization
20—170	variable preset

180—380	set character attributes
400—410	display current location
420—430	aboard rocket ship
440—480	check for objects present
650—660	health check
700—980	main command input and check
5000—5060	outside without spacesuit
5070—5090	character dead/lose game
5100—5170	directional moves
5180—5210	CRY
5300—5590	EAT
5700—5800	DRINK
6550—6590	PRAY
6810—7690	GET/DROP
9820—9840	WAIT
9850—9860	arms full
9900—9950	GET ALL
9960—9995	BLAST OFF

Subroutines

10000	timing delay loop
10010	instructions (incomplete)
10400—10440	score calculation
10450—10565	supplies present
10570—10690	junk items present
10700—10820	treasure items present
11150—11260	DIAGNOSIS
11300—11690	INVENTORY
11700—11780	error messages
11800—11870	HELP

Table 3.18 Variables Used in the "Mars" Program So Far.

AX	character's aim rating
DG	character's current health rating
DX	character's maximum health rating
G	GET or DROP
K	monster previously encountered
L1,L2	character's current location
L3,L4	character's previous location
Q	monster/obstacle present
PX	character's power rating
R1,R2	rocket ship location
SC	score
SV	monster kill score

SUMMARY

You now have a complete game of sorts, albeit a rather dull one at this point because it lacks real obstacles. In the next chapter programming will begin to liven up.

But before you add the monsters and other complications to the program, you should thoroughly test the partial program written so far. Try each of the commands several times. Enter bad commands to ensure that the error trapping routines work properly. At this point 17 commands are recognized. They are as follows:

N -- move north
S -- move south
E -- move east
W -- move west
SCORE -- display current score
DIA -- display diagnosis (report on current health rating)
HELP -- display list of possible commands
WAIT -- pause and increase current health rating
LOOK -- re-display current coordinates
GET xxx -- pick up an object (2 words required)
DROP xxx -- set down an object (2 words required)
INV -- INVENTORY objects currently being carried
EAT xxx -- 2 words required
DRINK
CRY
PRAY
BLAST OFF -- return to Earth; display final score and end the game

Once you are sure everything in the program so far works correctly, move on to the next chapter and complicate the poor explorer's life with monsters and geographic obstacles.

Chapter 4

Complicating the Game

The "heart and soul" of a good adventure game is the monsters and obstacles that plague the player in his quest.

This chapter adds disasters, setbacks, and even strokes of fortune to the MARS program begun in the last chapter. Here you will create creatures and landmarks.

MAPPING MONSTERS

The variable preset routine of Table 3.3 planted the obstacles in the main location map (LC(x,y)) with lines 40 to 60. Although the monsters and obstacles are in the program, the player can't see them yet.

First, add a display that will tell the player what obstacles, if any, he currently faces. In line 400, the variable Q was set to equal the value stored in the main map for the player's current location. Use the value of Q to determine whether anything is at the current map location.

A temporary routine for naming each obstacle as it is encountered is included in Table 4.1. Most of the lines will be changed as you move on to more advanced programming.

Add this routine to those from Chapter III and run a sample. All the obstacles should be scattered throughout the map area. Some locations will be blank. Be sure that no monsters appear in the rocketship's location.

Next add a map display routine as listed in Table 4.2, with the flow-chart shown in Figure 4.1.

Notice that the explored map (EX(x,y)) is the one displayed—not the complete map (LC(x,y)). Only those obstacles that the player has already encountered are displayed.

A typical map display is shown in Figure 4.2. Notice that the Marsquake, storm, and funny colored sky are not displayed. Neither is the ghost displayed on the map since each ghost appears only once, and then vanishes, leaving its space blank.

Table 4.1 Routine to Display the Obstacles in the "Mars" Program (Temporary Routine).

```
499  REM * CHECK FOR MONSTERS *
500  EX(L1,L2) = Q
510  IF Q = 1 PRINT "SQUEANLY SERPENT"
520  IF Q = 2 PRINT "GHOST"
530  IF Q = 3 PRINT "BRINCHLEY BEAST"
540  IF Q = 4 PRINT "KUFU"
550  IF Q = 5 PRINT "GRIMPH"
560  IF Q = 6 PRINT "PUROFOLEE"
570  IF Q = 7 PRINT "RIVER"
580  IF Q = 8 PRINT "MOUNTAIN"
590  IF Q = 9 PRINT "RAVINE"
600  IF Q = 10 PRINT "MARSQUAKE"
610  IF Q = 11 PRINT "STORM"
620  IF Q = 12 PRINT "FUNNY COLORED SKY"
```

Table 4.2 Adding the MAP Command to the "Mars" Program.

```
 950  IF QX$ = "MAP" GOSUB 12400; GOTO 700

12399  REM * MAP *
12400  CLS: PRINT: PRINT
12410  FOR X = 1 TO 10: PRINT "  ",: FOR Y = 1 TO 10
12415  IF L1 = X AND L2 = Y PRINT "+ ";: GOTO 12450
12420  Z = EX(X,Y):   IF Z < 1 OR Z > 9 GOTO 12440
12430  ON Z GOTO 12500, 12510, 12520, 12530, 12540, 12550, 12560, 12570,
       12580
12440  IF Z = 20 PRINT "* "; ELSE PRINT ".  ";
12450  NEXT: PRINT: NEXT: PRINT
12460  PRINT "+ = YOU,  * = SHIP, S = SQUEANLY SERPENT, B =
       BRINCHLEY BEAST,"
12470  PRINT "K = KUFU, G = GRIMPH, P = PUROFOLEE, R = RIVER, M =
       MOUNTAIN,"
12480  PRINT "r = RAVINE            PRESS 'ENTER' TO CONTINUE GAME  ";
12490  INPUT Q$: RETURN
12500  PRINT "S ";:GOTO 12450
12510  PRINT ".  ";: GOTO 12450
12520  PRINT "B ";: GOTO 12450
12530  PRINT "K ";: GOTO 12450
12540  PRINT "G ";: GOTO 12450
12550  PRINT "P ";: GOTO 12450
12560  PRINT "R ";: GOTO 12450
12570  PRINT "M ";: GOTO 12450
12580  PRINT "r ";: GOTO 12460
```

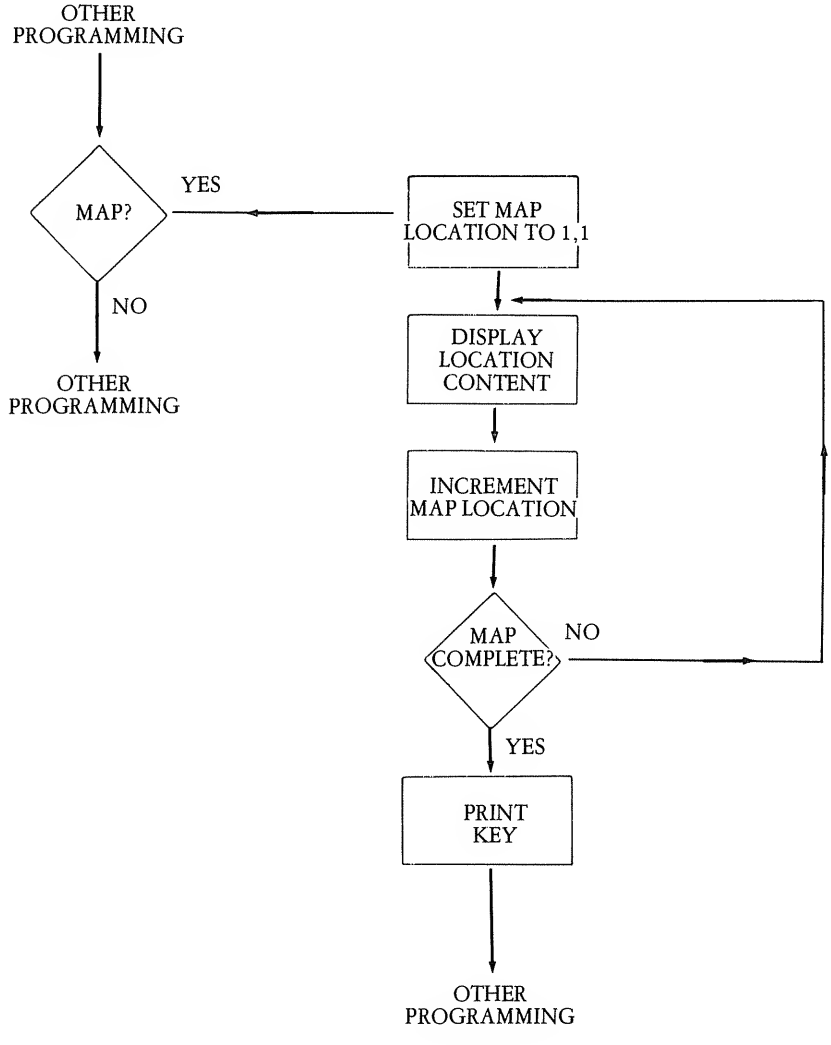


Figure 4.1 Flow-chart for MAP Routine.

SQUEANLY SERPENT

Just to wander about and be told monsters are present isn't too exciting. Later additions to the command sequence will allow the player more direct interaction with some of the obstacles.

For now, write expanded display routines so each type of monster begins to show a unique personality.

Start with the Squeanly serpent, which appears when Q has a value of 1. First, change the identification line to call up a subroutine rather than just print out the creature's name:

```
510 IF Q = 1 GOSUB 10950
```

The Squeanly serpent is a fairly simple beast that bites the explorer who encounters him and that's that. Of course, this is detrimental to the character's current health rating.

A player will have no way to kill, or deal directly with a Squeanly serpent. When he encounteres one of these nasty reptiles, the only thing he can do is to move on to a new location. Table 4.3 lists the complete Squeanly serpent subroutine.

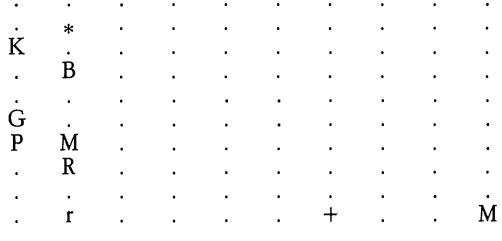


Figure 4.2 Typical Map for the Game of "Mars."

Table 4.3 Squeanly Serpent.

```
  510  IF Q = 1 GOSUB 10950

10949  REM * SERPENT *
10950  PRINT "DARN! You were just bitten by a Squeanly Serpent!"
10960  DG = DG - 5: RETURN
```

GHOST

At some locations (when Q = 2), a ghost of an ancient Martian appears:

```
520 IF Q = 2 GOSUB 11000
```

Sometimes the ghost is beneficial, and other times it will be neutral. There is no need to make everything in the game to the player's disadvantage.

Just a displayed message announces the ghost's appearance:

```
11000 PRINT "A ghost of an Ancient Martian suddenly appears
      before you!"
```

Then a pair of random coordinates are selected and the contents of that location is stored as Z:

```
11010 X = RND(10): Y = RND(10): Z = LC(X,Y)
```

If Z has a value of 1, 3, 4, 5, or 6 the ghost will announce its selected coordinates and tell what type of monster occupies that location:

```
11040 PRINT "'Lo,' sayith the ghost, 'at ";X;":";Y; "you shall find
      a ";
11050 IF Z = 1 PRINT "Squeanly serpent"
11060 IF Z = 3 PRINT "Brinchley Beast"
11070 IF Z = 4 PRINT "kufu"
11080 IF Z = 5 PRINT "Grimph"
11090 IF Z = 6 PRINT "purofolee"
```

When this information is recorded in the explored map array (EX(x,y)), the ghost vanishes, being erased from both the main and the explored maps. The subroutine ends here:

```
11100 EX(X,Y) = Z
11110 PRINT "The ghost vanishes into thin air!": LC(L1,L2) = 0:
      EX(L1,L2) = 0: RETURN
```

Since the ghost only reveals the location of certain types of obstacles, make some provision if the randomly selected coordinates do not contain one of these monsters.

If the selected location has a value greater than 6, a new set of coordinates will be selected:

```
11030 IF Z>6 GOTO 11010
```

The map location may also take a value of 2, or 0 (or negative numbers, which will be introduced later). If these values are encountered, you can loop around to select a new set of coordinates. However, the computer may take a long time to select a valid set of coordinates, especially late in a game when many of the monsters are dead. You need an alternative routine to prevent the program from latching up.

If the selected coordinates are the location of another ghost (Z = 2), or nothing (Z = 0) (or a dead monster—z < 0), the ghost will just say "BOO!" and vanish:

```
11020 IF Z<1 OR Z = 2 GOTO 11120
11120 PRINT "  ","BOO!": PRINT: GOTO 1110
```

Complete programming for the ghost is listed in Table 4.4. A flow-chart is shown in Figure 4.3.

As with the Squeanly serpent, there is no way the player can interact with a ghost. The ghost appears, says its bit, then vanishes permanently from the game. You may encounter more than one ghost.

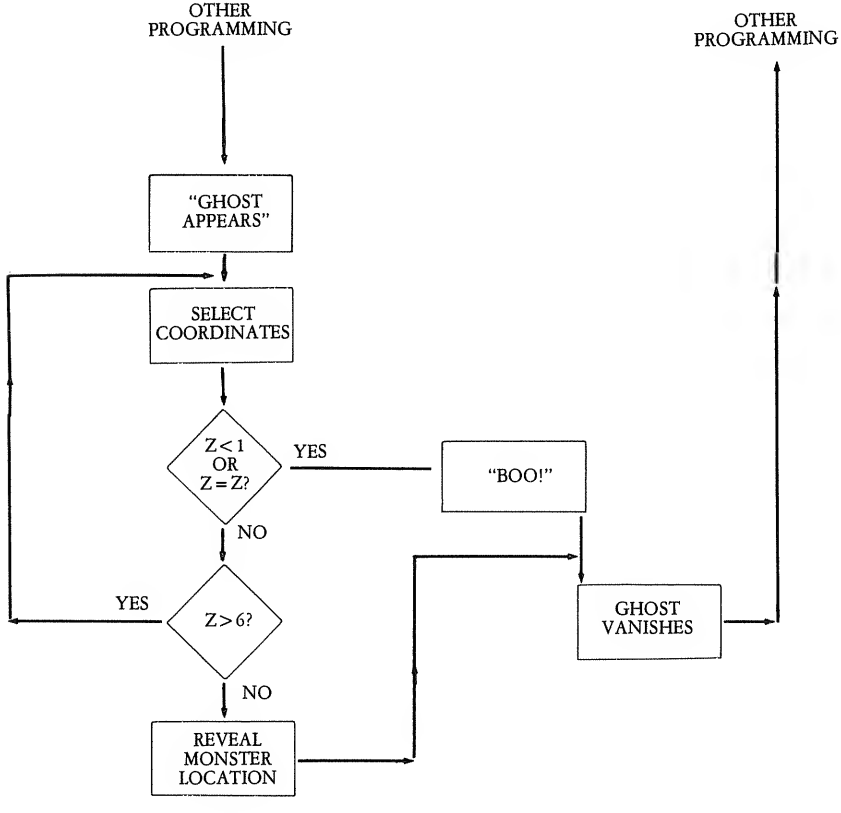


Figure 4.3 Flow-chart for GHOST Routine.

BRINCHLEY BEAST

The Brinchley Beast is the first monster with which the player can interact and fight:

530 **IF** Q = 3 **GOTO** 9300

When facing a Brinchley Beast, certain commands will have special results that will be dealt with later. For now, concentrate on the initial encounter.

Table 4.4 Ghost.

```
  520  IF Q = 2 GOSUB 11000

10999  REM * GHOST *
11000  PRINT "A ghost of an ancient Martian suddenly appears before you!"
11010  X = RND(10): Y = RND(10): Z = LC(X,Y)
11020  IF Z < 1 OR Z = 2 GOTO 11120
11030  IF Z > 6 GOTO 11010
11040  PRINT "'Lo,' sayeth the ghost, 'at ";X;":";Y;" you shall find a ";
11050  IF Z = 1 PRINT "Squeanly Serpent"
11060  IF Z = 3 PRINT "Brinchley Beast"
11070  IF Z = 4 PRINT "kufu"
11080  IF Z = 5 PRINT "Grimph"
11090  IF Z = 6 PRINT "purofolee"
11100  EX(X,Y) = Z
11110  PRINT "The ghost vanishes into thin air!": LC(L1,L2) = 0: EX(L1,L2) = 0:
       RETURN
11120  PRINT " ","BOO!": PRINT: GOTO 11110
```

One way to give your game creatures personality is to simulate their cries on the display screen. The Brinchley Beast gets its name from its distinctive cry of "brinch-LEY!!!"

The cry will be displayed only once when the creature is first encountered. The variable K (which is set to 0 on each location move) will be given a value of 1 to indicate the Beast has already been encountered. When K equals 1, the cry is skipped, and just a message stating that a Brinchley Beast is there will be displayed.

The first time a player encounters a Brinchley Beast, another variable (TB) is set to 0. This variable indicates whether or not the explorer has touched the Beast. This is important when you add the special commands later.

Table 4.5 shows the complete encounter routine for a Brinchley Beast.

Table 4.5 Brinchley Beast.

```
  530  IF Q = 3 GOTO 9300

 9300  IF K = 1 GOTO 9330
 9310  K = 1: TB = 0: PRINT " ","brinch-";: GOSUB 10000
 9320  PRINT "LEY!!!": PRINT: GOSUB 10000
 9330  PRINT "A Brinchley Beast is here.": GOTO 650
```

KUFU

The next monster encountered (Q = 4) is the kufu, a much nastier creature than the Brinchley Breast as indicated by the message which is displayed when a kufu is first encountered:

A hungry kufu blocks your path!

It salivates in foul anticipation of the meal to come (lines 9360 through 9380).

The variable K, set to a value of 2, indicates that this kufu has already been encountered. On subsequent encounters, a different message is displayed:

```
9350 IF K = 2 GOTO 9400
9400 PRINT "A kufu is here."
```

In either case, a random number from one to 10 is selected. If this number is greater than 6 (a 40 percent chance), the kufu will attack, biting the explorer in one of seven randomly selected places. Depending on where the character is bitten, points are subtracted from his current health rating (DG). A neck bite is much more serious than an arm bite. If the kufu bites the explorer's air hose, the character dies and the game ends:

```
9410 X = RND(10): IF X > 6 GOTO 6670
6670 PRINT "It bites your  ";: X = RND(7)
6680 IF X = 1 PRINT "arm!": DG = DG - 5
6690 IF X = 2 PRINT "leg!": DG = DG - 7
6700 IF X = 3 PRINT "stomach!": DG = DG - 50
6710 IF X = 4 PRINT "neck!": DG = DG - 75
6720 IF X = 5 PRINT "air hose!": GOTO 5070
6730 IF X = 6 PRINT "nose!": DG = DG - 20
6740 IF X = 7 PRINT "posterior!": DG = DG - 30
```

Whether or not the kufu attacks, program control is jumped back to line 650 for a health check. Even a bite on the arm can be fatal if the character is in weak enough condition (DG < 5).

Soon you will add commands for dealing with the kufu. Table 4.6 lists the programming for the kufu encounter.

GRIMPH

The next monster (Q = 5) listed in Table 4.7 is called a Grimph, the nastiest monster in the game of "MARS". When we add new commands, the Grimph will be very difficult to fight or escape from.

Table 4.6 Kufu.

```
 540  IF Q = 4 GOTO 9350

9350  IF K = 2 GOTO 9400
9360  PRINT "A hungry kufu blocks your path!": GOSUB 10000
9370  PRINT: PRINT "It salivates in foul anticipation of the meal to come!":
      PRINT
9380  K = 2: GOTO 9410
9400  PRINT "  A kufu is here."
9410  X = RND(10): IF X > 6 GOTO 6670
9420  GOTO 650

6670  PRINT "It bites your ";: X = RND(7)
6680  IF X = 1 PRINT "arm!": DG = DG - 5
6690  IF X = 2 PRINT "leg!": DG = DG - 7
6700  IF X = 3 PRINT "stomach!": DG = DG - 50
6710  IF X = 4 PRINT "neck!": DG = DG - 75
6720  IF X = 5 PRINT "air hose!": GOTO 5070
6730  IF X = 6 PRINT "nose!": DG = DG - 20
6740  IF X = 7 PRINT "posterior!": DG = DG - 30
6750  GOTO 650
```

Table 4.7 Grimph.

```
 550  IF Q = 5 GOTO 9450

9110  PRINT "THE GRIMPH BREAKS YOUR ";
9120  X = RND(10); IF X = 1 AND S(11) = 0 GOTO 9120
9130  IF X = 1 PRINT "COMPASS!": S(11) = 0
9140  IF X = 2 PRINT "ARM!": DG = DG - 15
9150  IF X = 3 PRINT "LEG!": DG = DG - 20
9160  IF X = 4 PRINT "NECK!": DG = DG - 70
9170  IF X = 5 PRINT "THUMBNAIL!": DG = DG - 2
9180  IF X = 6 PRINT "NOSE!": DG = DG - 10
9190  IF X = 7 PRINT "BIG TOE!": DG = DG - 3
9200  IF X = 8 PRINT "BACK!": DG = DG - 65
9210  IF X = 9 PRINT "FINGER!": DG = DG - 3
9220  IF X = 10 PRINT "SKULL!": DG = DG - 75
9230  GOTO 650

9450  PRINT "A Grimph is here.":X = RND (10); IF X > 3 GOTO 9110
9460  GOTO 650
```

When a Grimph is encountered, there is a 70 percent chance that it will attack:

```
9450 PRINT "A Grimph is here.": X = RND(10): IF X>3 GOTO
     9110
```

A Grimph, when it attacks, will break either some part of the explorer's anatomy, whether it be a thumbnail or a skull. It might also break the compass which is necessary for the current coordinates to be displayed.

Obviously, the character must be carrying the compass in order for the Grimph to break it, so an extra **IF**. . . **THEN**. . . statement is added so that if the compass is selected, but is not there (S(11) = 0) a new attack number will be selected:

```
9120 X = RND(10): IF X = 1 AND S(11) = 0 GOTO 9120
```

With or without an attack, an encounter with a Grimph ends with a health check (line 650).

PUROFOLEE

The purofolee is the last of the monsters a player can encounter. This creature appears when Q has a value of 6. Table 4.8 lists the programming for an encounter. K is set to a value of 3 after the first encounter.

On the first encounter, the purofolee's cry of "Gibble! Gibble! Gibble!" is heard (displayed) followed by "A wild purofolee hops into view."

On later encounters the computer simply reminds you that "A purofolee is here." and the purofolee inquires about your intentions:

"PUROFOLEE: Gibble?"

As with other monsters, several new commands will soon be added to the program for dealing with a purofolee.

Table 4.8 Purofolee.

```
560  IF Q = 6 GOTO 9500

9500 IF K = 3 GOTO 9550
9510 GOSUB 10000: FOR X = 1 TO 3: Z = RND(25) + 1: FOR Y = 1 TO Z
9520 PRINT " ";: NEXT Y: PRINT "gibble!  ";: NEXT X
9530 PRINT: PRINT: PRINT "A wild purofolee hops into view!": K = 3
9540 GOTO 650
9550 PRINT "A purofolee is here.": PRINT
9560 PRINT "PUROFOLEE: Gibble?": PRINT
9570 GOTO 650
```

RIVER

Now add natural obstacles or landmarks to the array of monsters. The first of these is a river (Q = 7), programmed in Table 4.9.

When K is set to 4, it indicates that the river has already been encountered. On the first encounter, RV is set to a randomly selected value from 1 to 5. RV values of 2, 4, or 5 have no special meaning. However, if RV = 1 the computer will tell you:

"It is quite pleasant here."

On the other hand, if RV = 3 you learn that:

The smell of ancient sewage is unpleasant, but bearable."

The character's current health rating (DG) benefits slightly by being on a river bank:

```
9680 DG = DG + .25: GOTO 650
```

Special commands for the RIVER will be added later.

Table 4.9 River.

```
 570  IF Q = 7 GOTO 9600

9600  IF K = 4 GOTO 9650
9610  RV = RND(5): PRINT "You come to a river bank"
9620  PRINT: K = 4: GOTO 9660
9650  PRINT "You are at the bank of a river.": PRINT
9660  IF RV = 1 PRINT "It is quite pleasant here."
9670  IF RV = 3 PRINT "The smell of ancient sewage is unpleasant, but bearable."
9680  DG = DG + .25: GOTO 650
```

MOUNTAIN/RAVINE

The next two obstacles, both dealt with in Table 4.10, are the MOUNTAIN (Q = 8) and the RAVINE (Q = 9).

While special commands will be added later, for now a simple message will be displayed when a mountain is encountered:

```
580 IF Q = 8 PRINT "You are at the foot of a tall, craggy mountain."
```

On the other hand, encounter of a ravine calls up a subroutine:

```
590 IF Q = 9 GOSUB 11900
```

Because of the nature of the ravine subroutine, no commands will be given to deal directly with this particular obstacle.

If the explorer comes across a ravine, he falls in and damages his current health rating (DG) by a random amount. He may also drop some of the items he is carrying. The items will be relocated randomly in nearby locations. He will always crawl out of the ravine to the south, ending up in a new map location. For this reason there are no special commands for the ravine.

Table 4.10 Mountain/Ravine.

```
  580  IF Q = 8 PRINT "You are at the foot of a tall, craggy mountain."
  590  IF Q = 9 GOSUB 11900

11899  REM * RAVINE *
11900  L3 = L1; L4 = L2; PRINT "YOU JUST FELL INTO A DEEP RAVINE!":
       DG = DG - RND(DG)
11910  FOR X = 1 TO 4: Y = RND(11): IF T(Y) = 1 GOTO 11980
11920  IF J(Y) = 1 GOTO 11990
11930  IF S(Y) > 0 GOTO 12000
11940  NEXT: PRINT: GOSUB 10000
11950  PRINT "It takes quite a bit of effort, but you manage to crawl out to "
11960  PRINT "the south.": PRINT: L1 = L1 + 1: IF L1 > 10 THEN L1 = 1
11970  Q = LC(L1,L2): EX(L1,L2) = Q: RETURN
11980  T(Y) = 0: E(Y) = RND(10): F(Y) = RND(10): GOTO 11940
11990  J(Y) = 0: C(Y) = RND(10): D(Y) = RND(10): GOTO 11940
12000  S(Y) = 0: A(Y) = RND(10): B(Y) = RND(10): GOTO 11940
```

MARSQUAKE

A Marsquake is the next obstacle. This is like an earthquake on Earth. A subroutine is called to create the action of the Marsquake:

```
600 IF Q = 10 GOSUB 12050
```

The complete subroutine is listed in Table 4.11. No commands will affect the Marsquake.

When the player encounters a Marsquake, a warning message will be displayed:

```
12050 PRINT "The ground begins to rumble beneath your feet!":
      PRINT
```

The bouncing ground throws the explorer into a new map location. He may be moved from −2 to +2 spaces in either, or both, directions. Of course you must include checks for out of range locations:

```
12060 L1 = RND(5) + L1 - 3: IF L1 > 10 THEN L1 = 1
12070 IF L1 < 1 THEN L1 = 10
```

```
12080 L3 = L1 - 1: IF L3 < 1 THEN L3 = 10
12090 L2 = L2 + RND(5) - 3: IF L2 < 1 THEN L2 = 10
12100 IF L2 > 10 THEN L2 = 1
12110 L4 = L2: FOR X = 1 TO 40
```

Next up to 40 map locations are randomly selected. If the location value is 20 (rocket ship), 0 (no occupant), or negative (dead monster) that location will be ignored. For other positive values, the polarity is reversed. That is, the values are made negative, killing the monster, or removing the geographic obstacles.

Note that the changes are made only on the main map array (LC(x,y)). The explored map array (EX(x,y)) is not automatically updated, so the explorer can trust his map just so far. It may contain errors after a Marsquake or certain other events.

If you prefer to make the game easier by automatically updating the explored map array (EX(x,y)), you can add this line:

```
12135 IF EX(Y,Z) > 0 THEN EX(Y,Z) = ZZ
```

Table 4.11 Marsquake.

```
  600 IF Q = 10 GOSUB 12050

12049 REM * MARSQUAKE *
12050 PRINT "The ground begins to rumble beneath your feet!": PRINT
12060 L1 = RND(5) + L1 - 3: IF L1 > 10 THEN L1 = 1
12070 IF L1 < 1 THEN L1 = 10
12080 L3 = L1 - 1: IF L3 < 1 THEN L3 = 10
12090 L2 = L2 + RND(5) - 3: IF L2 < 1 THEN L2 = 10
12100 IF L2 > 10 THEN L2 = 1
12110 L4 = L2: FOR X = 1 TO 40
12120 Y = RND(10): Z = RND(10): ZZ = LC(Y,Z): IF ZZ = 20 GOTO 12140
12130 IF ZZ > 0 THEN ZZ = -ZZ
12140 LC(Y,Z) = ZZ: NEXT
12150 PRINT " ","* whew! *": PRINT "The Marsquake is over now!"
12160 LC(L1,L2) = 0: Q = 0: EX(L1,L2) = 0: RETURN
```

STORM

The next obstacle (Q = 11) is a Martian storm. The storm is covered completely in a subroutine:

```
610 IF Q = 11 GOSUB 12200
```

No special commands will be offered, and storms will not be displayed on the explored map.

The storm subroutine begins by displaying an appropriate message, then the current location is cleared so that each storm will occur only once:

```
12200 PRINT "You are caught in a weird Martian storm!": PRINT
12210 LC(L1,L2) = 0: Q = 0: EX(L1,L2) = 0
```

The character's current health rating is then reduced by up to one fourth of its current value:

```
12220 DG = DG - RND(DG/4)
```

As in the Marsquake subroutine, up to 40 percent of the map location occupants are altered by the Martian storm. However, instead of just negating positive values, this subroutine makes negative values positive. Whatever the sign of the randomly selected location may be, its opposite replaces it. This means, for example, that dead monsters may be brought back to life.

Of course, the location containing a value of 20, the rocket ship's location, is protected:

```
12235 IF LC(Y,Z) = 20 GOTO 12250
```

By simply negating the current value, we can reverse its sign. For instance $-(-7) = +7$, or $-(+4) = -4$:

```
12240 LC(Y,Z) = -LC(Y,Z)
```

Once again, the changes brought about by the Martian storm are not indicated in the explored map array (EX(x,y)). If you want automatic map up-dating, add this line:

```
12245 EX(Y,Z) = -EX(Y,Z)
```

Notice that there is no need to check for unoccupied locations for bypassing. An unoccupied location is represented by a value of 0, and negating 0 has no effect ($-0 = 0$).

Table 4.12 lists the complete subroutine for the Martian storm.

Table 4.12 Storm.

```
  610 IF Q = 11 GOSUB 12200

12199 REM * STORM *
12200 PRINT "You are caught in a weird Martian storm!": PRINT
12210 LC(L1,L2) = 0: Q = 0: EX(L1,L2) = 0
12220 DG = DG - RND(DG/4)
```

```
12230  FOR X = 1 TO 40: Y = RND (10): Z = RND(10)
12235  IF LC(Y,Z) = 20 GOTO 12250
12240  LC(Y,Z) = -LC(Y,Z)
12250  NEXT: GOSUB 10000
12260  PRINT "The weather seems to be clearing up now.": PRINT
12270 RETURN
```

FUNNY COLORED SKY

When the player encounters the final obstacle (Q = 12), the sky changes color for awhile:

```
  620 IF Q = 12 GOSUB 12300
12300 PRINT "ODD . . . THE SKY TURNS A FUNNY COLOR FOR
      A FEW MINUTES . . ."
12310 PRINT: GOSUB 10000
```

(Subroutine 10000 is the time delay subroutine introduced in Chapter 3.)

Table 4.13 lists the complete funny colored sky subroutine. Figure 4.4 outlines the flow-chart.

When the sky above Mars changes color, some very strange events take place. The obstacles at up to 25 map locations change into other obstacles. This is done by adding 1 to each of the selected obstacle values:

```
12320 FOR X = 1 TO 25: Y = RND(10): Z = RND(10)
12330 ZZ = LC(Y,Z): IF ZZ = 20 GOTO 12360
12340 LC(Y,Z) = ZZ + 1
12360 NEXT: PRINT "Well, everything seems to be back to normal
      now. "
```

Once again, the special value 20 is protected so the rocket ship will not change.

Table 4.14 shows some of the changes which may be made. The changes are not reflected in the explored map array (EX(x,y)).

One additional protection line will avoid problems. For example, a specific location starts out with a value of 12. If this location is selected when the player encounters a "funny colored sky", this value will be raised by 1 to 13. This is an undefined value and will behave no differently from a blank space. There is no problem so far.

But suppose, in the course of a long game, that the same space is selected by "funny colored sky" subroutines 7 more times. This would certainly not be impossible. The location would now have a value of 20, the value

reserved for the rocket ship. Two rocket ships would now be displayed on the player's map. To prevent this, we can add this line:

```
12335 IF ZZ>12 THEN ZZ = -1
```

Why is the value set to −1, rather than 0? This is because when 1 is added to the new value, the result will be 0, or a blank space.

You might prefer to let "phantom rocket ships" appear on the map. Since the active location is determined by the values of R1 and R2, only the original rocket ship location will be functional. You can't BLAST OFF, for example, unless you are aboard the real ship, and the game could get complicated if you fail to memorize your ship's original location.

Table 4.13 Funny Colored Sky.

```
  620  IF Q=12 GOSUB 12300

12299  REM * SKY *
12300  PRINT "ODD... THE SKY TURNS A FUNNY COLOR FOR A FEW
       MINUTES..."
12310  PRINT: GOSUB 10000
12320  FOR X= 1 TO 25: Y= RND(10): Z=RND(10)
12330  ZZ=LC(Y,Z): IF ZZ=20 GOTO 12360
12335  IF ZZ>12 THEN ZZ= -1
12340  LC(Y,Z)= ZZ + 1
12360  NEXT: PRINT "Well, everything seems to be back to normal now. ";
12370  GOSUB 10000: PRINT "I guess...": PRINT
12380  RETURN
```

Table 4.14 Sample Effects of a "Funny Colored Sky".

Old Value		New Value	
0	blank	1	Squeanly Serpent
1	Squeanly Serpent	2	Ghost
5	Grimph	6	Purofolee
9	Ravine	10	Marsquake
12	funny colored sky	13	blank
−7	blank	−6	dead Purofolee
−1	dead Squeanly Serpent	0	blank

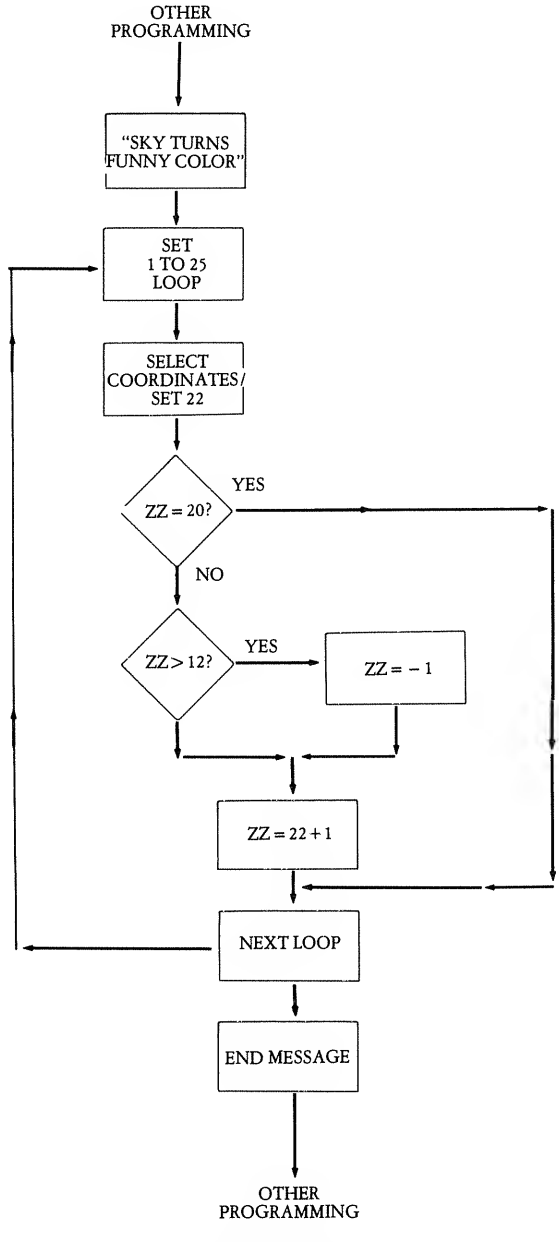


Figure 4.4 Flow-chart for FUNNY COLORED SKY Routine.

DEAD MONSTERS

Basic routines for each of the 12 obstacles are now added. Before adding commands to deal with some of these obstacles, add a simple subroutine to display dead monsters. As mentioned earlier, a dead monster is indicated by a negative number; for instance, −3 is a dead Brinchley Beast. Simply set up some **IF...THEN...** tests to display the appropriate message, as shown in Table 4.15.

Except for the **PRINT** "here." statement, line 10920 is not essential. It simply prints out an extra message; i.e., "The smell is horrendous", under certain conditions. This message will always be displayed for dead Brinchley Beasts (Q = −3) and dead Grimphs (Q = −5). There is a 30 percent chance that it will be printed for other dead monsters.

Table 4.15 Dead Monsters.

```
  500  EX(L1,L2)=Q: IF (Q<0) AND (Q>-7) GOSUB 10850

10849  REM * DEAD MONSTER *
10850  PRINT "There is a dead  ";
10860  IF Q = -1 PRINT "Squeanly serpent";
10870  IF Q= -2 PRINT "ghost of an Ancient Martian";
10880  IF Q = -3 PRINT "Brinchley Beast";
10890  IF Q = -4 PRINT "kufu";
10900  IF Q = -5 PRINT "Grimph";
10910  IF Q = -6 PRINT "purofolee";
10920  PRINT "here.": X=RND(10): IF (X>7) OR (Q = -3) OR (Q = -5)
       PRINT "The smell is horrendous!"
10930  RETURN
```

MOVING PAST MONSTERS

An obstacle by definition blocks one's path. The monsters should not stand passively by and let the explorer freely use the regular "N", "S", "E", and "W" move commands. They should block as listed in Table 4.16.

First off, whenever the player enters a new location, a special subroutine is called:

```
  405 GOSUB 12600
12600 IF Q=3 THEN MM=RND(4)
12610 IF Q=4 THEN MM=RND(6)
12620 IF Q=5 THEN MM=RND(10)
12630 IF Q=6 THEN MM=RND(5)
12640 RETURN
```

A random value is assigned to the variable MM. The range of possible values is determined by which of the four monsters (Brinchley Beast—3, kufu—4, Grimph—5, or purofolee—6) is present. The program uses the variable MM only when Q equals 3, 4, 5, or 6. For other obstacles, it is irrelevant, and this subroutine simply wastes a few microseconds.

Next, you need to add a line in the main command sequence to check for the presence of potentially path blocking monsters. Q will have a value of 3, 4, 5, or 6:

```
715 IF (Q>2) AND (Q<7) GOTO 1000
```

Next check Q$ since the blocking routine is only relevant for directional move commands ("N", "S", "E", or "W"). If the command is not one of the four move commands, the program reverts back to the main command sequence:

```
1000 IF Q$ = "N" OR Q$ = "S" OR Q$ = "E" OR Q$ = "W" GOTO
     1010
1005 GOTO 760
```

Now compare the directional command with the value of MM. If MM equals 1, the monster will let the explorer move to the north. If MM equals 2, it will let the explorer move south. For an MM value of 3, it will let him move east. If MM has a value of 4, the monster will let the player move to the west. If MM has a value greater than 4, the monster will not let the player move at all.

Table 4.16 Moving Past Monsters.

```
 405  GOSUB 12600

 715  IF (Q>2) AND (Q<7) GOTO 1000

 999  REM * MONSTER BLOCK MOVE *
1000  IF Q$ = "N" OR Q$ = "S" OR Q$ = "E" OR Q$ = "W" GOTO 1010
1005  GOTO 760
1010  IF Q$ = "N" AND MM = 1 GOTO 5100
1015  IF Q$ = "S" AND MM = 2 GOTO 5120
1020  IF Q$ = "E" AND MM = 3 GOTO 5140
1025  IF Q$ = "W" AND MM = 4 GOTO 5160
1030  IF Q = 5 GOTO 9100
1040  IF Q = 3 PRINT "THE BRINCHLEY BEAST BLOCKS YOUR PATH."
1050  IF Q = 4 PRINT "THE KUFU WILL NOT LET YOU GO THAT WAY.":
      DG = DG - 2
1060  IF Q = 6 PRINT "The durn purofolee is in your way!"
```

```
1070  GOTO 650

9100  PRINT "NONE TOO PLEASED WITH YOUR ATTITUDE,  ";

12600 IF Q = 3 THEN MM = RND(4)
12610 IF Q = 4 THEN MM = RND(6)
12620 IF Q = 5 THEN MM = RND(10)
12630 IF Q = 6 THEN MM = RND(5)
12640 RETURN
```

The Brinchley Beast is the most agreeable of the monsters since it will always allow the player to move in one of the four move directions (MM = **RND**(4)). The Grimph, on the other hand, is the most troublesome since 60 percent of the time it won't let the player move in any direction (MM = **RND**(10)).

If the command and MM value agree, an ordinary move is made:

```
1010 IF Q$ = "N" AND MM = 1 GOTO 5100
1015 IF Q$ = "S" AND MM = 2 GOTO 5120
1020 IF Q$ = "E" AND MM = 3 GOTO 5140
1025 IF Q$ = "W" AND MM = 4 GOTO 5160
```

If the player's attempted move is blocked, an appropriate message is displayed for each type of monster:

```
1040 IF Q = 3 PRINT "THE BRINCHLEY BEAST BLOCKS YOUR
     PATH."
1050 IF Q = 4 PRINT "THE KUFU WILL NOT LET YOU GO
     THAT WAY.": DG = DG - 2
1060 IF Q = 6 PRINT "The durn purofolee is in your way!"
```

Note that if the monster is a kufu, the current health rating (DG) will be decreased by 2.

The Grimph, the nastiest monster, expresses its displeasure at the explorer's attempted move by attacking:

```
1030 IF Q = 5 GOTO 9100
9100 PRINT "NONE TOO PLEASED WITH YOUR
     ATTITUDE,  ";
*9110 PRINT "THE GRIMPH BREAKS YOUR  ";
```

*The Grimph attack routine was listed in Table 4.7.

As the program is written here, if the player enters a "LOOK" command, a new value of MM will be selected.

MOVING PAST RIVERS AND MOUNTAINS

Rivers (Q = 7) and mountains (Q = 8) will also affect the directional move commands. If a player encounters one of these obstacles, he may only move back the way he came as defined by L3 and L4 (previous location coordinates). Table 4.17 lists the programming for this routine.

Table 4.17 Moving Past Rivers/Mountains.

```
 717  IF (Q = 7 OR Q = 8) AND (Q$ = "N" OR Q$ = "S" OR Q$ = "E" OR
      Q$ = "W") GOTO 9700

9700  X = L1: Y = L2: IF Q$ = "N" THEN X = L1 - 1
9710  IF X < 1 THEN X = 10
9720  IF Q$ = "S" THEN X = L1 + 1
9730  IF X > 10 THEN X = 1
9740  IF Q$ = "E" THEN Y = L2 - 1
9750  IF Y < 1 THEN Y = 10
9760  IF Q$ = "W" THEN Y = L2 + 1
9770  IF Y > 10 THEN Y = 1
9780  IF X = L3 AND Y = L4 GOTO 9810
9790  IF Q = 7 PRINT "THE RIVER "; ELSE PRINT "THE MOUNTAIN ";
9800  PRINT "IS IN YOUR WAY.": DG = DG - 3: GOTO 650
9810  L3 = L1: L4 = L2: L1 = X: L2 = Y: K = 0: GOTO 400
```

EXPANDING THE "CRY" COMMAND

The last chapter included the command "CRY", which resulted only in display of a sarcastic message. There may be other results from this action when certain monsters are encountered. Table 4.18 shows the lines required to expand the CRY command.

This command "**CRY**" will have special results when facing a kufu (Q = 4) and a purofolee (Q = 6).

If the player cries in front of a kufu, a special message is displayed:

```
5180 IF Q = 4 GOTO 5220
5220 PRINT "Kufus aren't noted for being soft-hearted."
5230 GOTO 500
```

CRYing in front of a purofolee badly disturbs this creature, and there is a 50 percent chance that it will attack:

```
5190 IF Q = 6 GOTO 5240
5240 X = RND(10): IF X > 5 GOTO 5270
```

```
5250 PRINT "Annoyed by the noise you're making, the purofolee
     knocks you": PRINT "down and jumps on your stomach 17
     times!"
5260 DG = DG - 17: GOTO 500
```

The other 50 percent of the time, CRYing will frighten the purofolee off to be randomly re-located in a near-by location. Line 5280 includes a check to prevent the original location from being re-selected.

Similarly, line 5285 prevents the rocket ship's location from being chosen as the purofolee's new location:

```
5270 PRINT "Frightened by the noise you're making, the purofolee
     hops off."
5280 X = L1 + RND(5) - 3: Y = L2 RPL RND(5) - 3: IF X = L1
     AND Y = L2 GOTO 5280
5285 IF X = R1 AND Y = R2 GOTO 5280
5290 LC(L1,L2) = 0: EX(L1, L2) = 0: Q = 0: LC(X,Y) = 6:
     GOTO 500
```

Note that any previous occupant of the selected location will be erased and replaced by the purofolee.

Table 4.18 Expanding the CRY Command.

```
 790 IF QX$ = "CRY" GOTO 5180

5179 REM * CRY *
5180 IF Q = 4 GOTO 5220
5190 IF Q = 6 GOTO 5240
5200 PRINT "Why? Are you upset for some reason?"
5210 GOTO 500
5220 PRINT "Kufus aren't noted for being soft-hearted."
5230 GOTO 500
5240 X = RND(10): IF X > 5 GOTO 5270
5250 PRINT "Annoyed by the noise you're making, the purofolee knocks you":
     PRINT "down and jumps on your stomach 17 times!"
5260 DG = DG - 17: GOTO 500
5270 PRINT "Frightened by the noise you're making the purofolee hops off."
5280 X = L1 + RND(5) - 3: Y = L2 + RND(5) - 3: IF X = L1 AND Y = L2
     GOTO 5280
5285 IF X = R1 AND Y = R2 GOTO 5280
5290 LC(L1,L2) = 0: EX(L1,L2) = 0: Q = 0: LC(X,Y) = 6: GOTO 500
```

EXPANDING THE "EAT" COMMAND

Table 4.19 shows a number of additions to the "EAT" command which was originally programmed in the last chapter.

The first step when an "EAT" command is recognized is to see if one of three monsters—kufu (Q = 4), Grimph (Q = 5), or purofolee (Q = 6)—is present. The Brinchley Beast, ghost, and Squeanly serpent will not respond to the explorer EATing:

```
5300 IF Q>3 AND Q<7 GOTO 5600
```

If the player enters an "**EAT**" command while facing a kufu, a warning message will be displayed, and his current health rating (DG) will be penalized two points:

```
5600 IF Q = 4 GOTO 5640
5640 PRINT "this is not time to think of your stomach,  "; N$;"!"
5650 GOSUB 10000: PRINT "Unless you want to see the inside of a
     kufu's stomach!"
5660 DG = DG - 2: GOTO 500
```

Subroutine 10000 is the time delay loop.

To **EAT** in front of a Grimph is a particularly bad idea. It thinks you are making a suggestion and eats you:

```
5610 IF Q = 5 GOTO 5670
5670 PRINT "The Grimph thinks that's a jim dandy idea!": PRINT:
     GOSUB 10000
5680 PRINT "It eats you!": PRINT: GOTO 5070
```

The character dead/end-of-game routine is at 5070. This was programmed at an earlier stage.

Table 4.19 Expanding the EAT Command.

```
5300  IF Q>3 AND Q<7 GOTO 5600
5320  IF QY$ = "UFU" GOTO 5480
5330  IF QY$ = "LEE" GOTO 5540

5480  IF Q = -4 GOTO 5500
5490  PRINT "First you have to go out and kill one, ";N$: DG = DG - 1:
      GOTO 500
5500  PRINT "You have to hold your nose to get close enough,"
5510  DG = DG + RND(15) - 10: IF DG>DX THEN DG = DX
```

```
5520 PRINT "but you somehow manage to get it down.": LC(L1,L2) = 0
5530 EX(L1,L2) = 0: Q = 0: GOTO 500
5540 IF Q = -6 GOTO 5560
5550 GOTO 5490
5560 PRINT " ","YUMMY!": PRINT: DG = DG + 50
5570 IF DG > DX THEN DG = DX
5580 LC(L1,L2) = 0: EX(L1,L2) = 0: 1 = 0: GOTO 500

5600 IF Q = 4 GOTO 5640
5610 IF Q = 5 GOTO 5670
5620 IF S(1) = 0:PRINT "There is nothing here to eat.": DG = DG - 1: GOTO
     500
5630 PRINT "When you take your food out, the durn purofolee snatches it"
5635 PRINT "before you can take a single bite!": PRINT: S(1) = 0: DG = DG -
     1: GOTO 500
5640 PRINT "This is no time to think of your stomach, ";N$;"!"
5650 GOSUB 10000: PRINT "Unless you want to see the inside of a kufu's
     stomach!"
5660 DG = DG - 2: GOTO 500
5670 PRINT "The Grimph thinks that's a jim dandy idea!": PRINT: GOSUB
     10000
5680 PRINT "It eats you!": PRINT: GOTO 5070
```

If you try to EAT in front of a purofolee when you don't have any food, you will be reminded that there is nothing available to eat:

```
5620 IF S(1) = 0 PRINT "There is nothing here to eat.": DG = DG -
     1: GOTO 500
```

However, if the explorer is carrying the food, he won't be for long. Purofolees, natural thieves that they are, will steal the food before the character can eat it:

```
5360 PRINT "When you take your food out, the durn purofolee
     snatches it"
5640 PRINT "before you can take a single bite!": PRINT: S(1) = 0:
     DG = DG - 1: GOTO 500
```

So far, the food from the supplies is the only thing the character can eat. If you add a few more lines, the player can make a feast of certain dead monsters. Only kufus and purofolees will be recognized as edible.

Start with the kufu:

```
5320 IF QY$ = "UFU" GOTO 5480
```

Naturally, a dead kufu must be present in order for the explorer to eat it. A dead kufu is represented by a Q value of −4. Check for it like this:

```
5480 IF Q = -4 GOTO 5500
5490 PRINT "First you have to go out and kill one,  ";N$: DG =
     DG - 1: GOTO 500
```

A kufu is not the tastiest thing on Mars. Lines 5500 and 5520 inform you that:

> You have to hold your nose to get close enough, but you somehow manage to get it down.

Lines 5520 and 5530 also clear the map array location. After all, once the kufu is eaten, it's carcass shouldn't still be lying there.

EATing a dead kufu is an unreliable source of nourishment. It may increase the current health rating by up to 5, or it may decrease it up to 10 points:

```
5510 DG = DG + RND(15) - 10: IF DG>DX THEN DG = DX.
```

A much better meal may be made from a dead purofolee:

```
5330 IF QY$ = "LEE" GOTO 5540
```

Once again, a dead purofolee (Q = −6) must be present for the explorer to eat it:

```
5540 IF Q = -6 GOTO 5560
5550 GOTO 5490
```

If a dead purofolee is available, **EAT**ing it proves to be both delicious and nutritious, adding 50 points to the current health rating (DG):

```
5560 PRINT"  ","YUMMY!": PRINT: DG = DG + 50
```

Of course the current health rating (DG) is not permitted to exceed the maximum health rating (DX):

```
5570 IF DG>DX THEN DG = DX
```

Once the purofolee is eaten, the map location is cleared to get rid of the body:

```
5580 LC(L1, L2) = 0: EX(L1,L2) = 0: Q = 0: GOTO 500
```

EXPANDING THE "DRINK" COMMAND

It is not reasonable for an explorer to **DRINK** from his BOTTLE OF WATER when he is at a RIVER (Q = 7). Table 4.20 includes programming that lets the explorer drink directly from the RIVER when appropriate:

```
5700 IF Q = 7 GOTO 5810
5810 PRINT"  ","slurp!!!": PRINT: GOSUB 10000
```

Ordinarily, drinking from a river improves the character's current health rating by 15 percent of the maximum value, provided that the current rating (DG) does not exceed the maximum rating (DX):

```
5840 DG = DG + DX * .15: IF DG>DX THEN DG = DX
5850 GOTO 500
```

To add a little variety, a random number from 1 to 10 is selected by the computer. If this number is greater than 7 (30 percent chance), the water is particularly good. Besides being tasty, it adds an extra 10 percent of the maximum health rating (DX) to the current health rating (DG) for a total grain of up to 25 percent of DX:

```
5820 X = RND(10): IF X>7 GOTO 5860
5860 PRINT "It sure tastes good!": DG = DG + DX * .1: GOTO
     5840
```

Notice that the computer does not have to do the DG > DX test here since it will be performed in line 5840 anyway.

If the randomly selected value (X) is less than 4 (again, a 30 percent chance), the water in the river turns out to be polluted and drops the current health rating (DG) by 25 percent, leaving it at 75 percent of its original level:

```
5830 IF X<4 GOTO 5870
5870 PRINT "This water makes you quite ill.": DG = DG * .75
5880 PRINT "But you don't die  ";:GOSUB 10000
5890 PRINT "—at least, not yet.": GOTO 500
```

There is a 40 percent chance (X = 4, 5, 6, or 7) that the river will contain just ordinary water.

Table 4.20 Expanding the DRINK Command.

```
5700 IF Q = 7 GOTO 5810

5810  PRINT"," "slurp!!!": PRINT: GOSUB 10000
5820  X = RND(10): IF X>7 GOTO 5860
```

```
5830  IF X<4 GOTO 5870
5840  DG = DG + DX * .15: IF DG>DX THEN DG = DX
5850  GOTO 500
5860  PRINT "It sure tastes good!": DG = DG + DX * .1: GOTO 5840
5870  PRINT "This water makes you quite ill.": DG = DG * .75
5880  PRINT "But you don't die ";: GOSUB 10000
5890  PRINT "—at least, not just yet.": GOTO 500
```

ADDING THE "FILL" COMMAND

As mentioned earlier, the player can refill his water bottle at a river. To do so he enters the "FILL" command. Table 4.21 lists the programming for this new command that will only be recognized when the explorer is at a river (Q = 7):

```
820 IF Q = 7 AND QX$ = "FIL" GOTO 5900
```

The player must specify an object to be filled with a command that is at least 6 characters long or no object will be specified and an error message is displayed:

```
5900 X = LEN(Q$): IF X>6 GOTO 5920
5910 PRINT "Fill what?": DG = DG - .5: GOTO 500
```

The string variable QY$ has already been set to equal the last three characters of the complete command. Use this substring to determine which object is specified.

The bottle (QY$ = "TLE") is the logical choice:

```
5920 IF QY$ = "TLE" GOTO 5990
```

The computer must make two checks before the bottle can be filled. First, the explorer must be carrying it; and second, it must be empty. If the bottle is empty S(2) will hold a value of 2. You only need to change this to 1 (to indicate a full bottle) and display an appropriate message:

```
5990 IF S(2) = 2 GOTO 6030
6030 PRINT "OK. The bottle is now full.": S(2)=1: GOTO 500
```

If the bottle is already full (S(2) = 1), the player cannot fill it again:

```
6000 IF S(2)=1 GOTO 6020
6020 PRINT "It's already full": DG = DG - 1: GOTO 500
```

If S(2) fails both of these tests (has a value that is neither 1 nor 2), it is assumed to equal 0, meaning the character does not have the bottle. You can't fill a bottle you don't have:

```
6010 PRINT "YOU DON'T HAVE IT,  ";N$;"!": DG = DG -
     1.25: GOTO 500
```

In preliminary testing of this game, players often tried to **FILL** some of the other objects when the BOTTLE was not available. These objects included the URN, the HUMMING BOX, the COPPER BOWL, and the SILVER CUP. One player even tried to fill his SPACESUIT. So for a nice touch, the program recognizes these objects and responds accordingly. Trying to **FILL** these objects will produce special error messages and, in some cases, problems for the player.

A universal error message should also be included, in case a player comes up with a possibility you didn't think of while programming the game:

```
5980 PRINT "THAT WON'T HOLD WATER,  ";N$: DG = DG
     - 1; GOTO 500
```

For the URN, COPPER BOWL, and SILVER CUP, just print a message that they leak and subtract .75 from the current health rating (DG). You must also include checks to make sure the character is actually carrying the specified object:

```
5930 IF QY$ = "URN" GOTO 6040
5960 IF QY$ = "OWL" GOTO 6080
5970 IF QY$ = "CUP" GOTO 6090
6040 IF J(8) = 0 GOTO 6010 ELSE PRINT "The urn";
6050 PRINT " leaks.": DG = DG - .75: GOTO 500
6080 IF T(1) = 0 GOTO 6010 ELSE PRINT "The copper bowl";:
     GOTO 6050
6090 IF T(5) = 0 GOTO 6010 ELSE PRINT "The silver cup";: GOTO
     6050
```

The humming box (T(9)) is a special object. An attempt to fill it with water will be penalized, and the box will dissolve to be permanently lost:

```
5940 IF QY$ = "BOX" GOTO 6060
6060 IF T(9) = 0 GOTO 6010
6070 PRINT "The box dissolves.": T(9) = 0: GOTO 500
```

As for the player who tried to fill his spacesuit with water, a drastic penalty awaits:

```
5950 IF QY$ = "UIT" PRINT "You drown!": GOTO 5070
```

The attempt is fatal.

Table 4.21 Fill.

```
820  IF Q = 7 AND QX$ = "FIL" GOTO 5900

5899 REM * FILL *
5900 X = LEN(Q$): IF X>6 GOTO 5920
5910 PRINT "Fill what? ": DG = DG - .5: GOTO 500
5920 IF QY$ = "TLE" GOTO 5990
5930 IF QY$ = "URN" GOTO 6040
5940 IF QY$ = "BOX" GOTO 6060
5950 IF QY$ = "UIT" PRINT "You drown!": GOTO 5070
5960 IF QY$ = "OWL" GOTO 6080
5970 IF QY$ = "CUP" GOTO 6090
5980 PRINT "THAT WON'T HOLD WATER, ";N$: DG = DG - 1: GOTO
     500
5990 IF S(2) = 2 GOTO 6030
6000 IF S(2) = 1 GOTO 6020
6010 PRINT "YOU DON'T HAVE IT,";N$;"!": DG = DG - 1.25: GOTO 500
6020 PRINT "It's already full": DG = DG - 1: GOTO 500
6030 PRINT "OK. The bottle is now full.": S(2) = 1: GOTO 500
6040 IF J(8) = 0 GOTO 6010 ELSE PRINT "The urn";
6050 PRINT" leaks.": DG = DG - .75: GOTO 500
6060 IF T(9) = 0 GOTO 6010
6070 PRINT "The box dissolves.": T(9) = 0: GOTO 500
6080 IF T(1) = 0 GOTO 6010 ELSE PRINT "The copper bowl";:GOTO 6050
6090 IF T(5) = 0 GOTO 6010 ELSE PRINT "The silver cup";: GOTO 6050
```

ADDING "INFLATE" COMMAND

Since one of the supplies is an inflatable raft (S(7)), an "INFLATE" command is appropriate. Table 4.22 lists the programming for this new command.

The **INFLATE** command will only be recognized by the program when the explorer is carrying the raft:

```
830 IF S(7) = 1 AND QX$ = "INF" GOTO 6100
```

Of course, it only makes sense to inflate your raft when you are at a river (Q = 7), so something should happen if a player enters the command at some other time. To penalize the player for this illogical command, have the raft spring a leak and become useless:

```
6110 GOSUB 10000: IF Q = 7 GOTO 6130
6120 PRINT "The raft springs a leak!": PRINT: S(7) = 0: GOTO 500
```

The player can **INFLATE** his RAFT to cross a RIVER, but you don't want things to be too easy. The river crossing will decrease the current health rating (DG), the raft will be lost, and the player will have no way to steer.

His new location will be randomly determined (line 6170).

Table 4.22 Inflate.

```
 830  IF S(7) = 1 AND QX$ = "INF" GOTO 6100

6099  REM * INFLATE RAFT *
6100  PRINT" ","Phfffftt!": PRINT
6110  GOSUB 10000: IF Q = 7 GOTO 6130
6120  PRINT "The raft springs a leak!": PRINT: S(7) = 0: GOTO 500
6130  DG = DG - (RND (50)/10)
6140  PRINT "You cannot control the raft on these raging currents"
6150  PRINT: GOSUB 10000
6160  PRINT "You make it to a shore, but you lose your raft!"
6170  X = L1 - 2 + RND(3): Y = L2 - 2 + RND(3): IF X = L1 AND Y = L2
      GOTO 6170
6172  IF X<1 THEN X = 10
6174  IF X>10 THEN X = 1
6176  IF Y<1 THEN Y = 10
6178  IF Y>10 THAN Y = 1
6180  S(7) = 0: L3 = L1: L4 = L2: L1 = X: L2 = Y: INPUT "Please press
      'ENTER' ";Q$
6190  GOTO 400
```

ADDING "CLIMB" COMMAND

When facing a mountain (Q = 8), the player can only move back the way he came (as determined by L3 and L4). Sometimes he may have to get past a mountain in another direction. To do this, add a new command—"**CLIMB**". This command will only be recognized when there is a mountain present:

```
850 IF Q = 8 AND QX$ = "CLI" GOTO 6200
```

When the **CLIMB** command is entered, the player is asked which way he wants to go. He must enter one of the four directional commands ("**N**", "**S**", "**E**", or "**W**"):

```
6200 INPUT "DIRECTION";Q$: Q$ = LEFT$(Q$,1)
6210 IF (Q$ = "N") OR (Q$ = "S") OR (Q$ = "E") OR (Q$ = "W")
     GOTO 6230
6220 GOTO 6200
```

You might want to include a way to let the player change his mind.

Adding the following line allows the player to enter "X" to cancel the **CLIMB** command:

```
6205 IF Q$ = "X" GOTO 700
```

After the player enters the desired direction, the computer checks to see if the coil of rope (S(6)) is being carried:

```
6230 IF S(6) = 1 GOTO 6270
```

For now we will assume that the character does not have the rope. He selects a random number from 1 to 10:

```
6240 X = RND(10)
```

Then the computer checks to see if the explorer is carrying the slimy thing (J(12)):

```
6250 IF J(12) = 1 GOTO 6320
```

If the player does not have the slimy thing, there is a 30 percent chance that he will successfully climb the mountain (program control jumps back to line 720 where the main control sequence makes a normal directional move):

```
6260 IF X > 7 GOTO 720 ELSE GOTO 6360
```

There is a 70 percent chance that the explorer will fall off the side of the mountain and reduce his current health rating (DG) by a randomly determined amount:

```
6360 PRINT "You fall!"
6370 DG = DG - RND(DG) + 1
6380 GOTO 500
```

If the explorer is carrying the slimy thing, but not the rope, there is a 33 percent chance of the above catastrophe:

```
6320 Y = RND(12): IF Y < 5 GOTO 6360
```

There is a 66 percent chance that the explorer will drop the slimy thing while part way up the mountain:

```
6330 IF S(12) = 0 GOTO 6360 ELSE PRINT "Half way up the moun-
     tain, you drop the slimy thing!"
```

There is now about a 57 percent chance that the character will slip on the dropped slimy thing and fall off of the mountain. Otherwise, it reverts

back to line 6260 for the regular no rope chance of falling:

```
6340 J(12) = 0: C(12) = L1: D(12) = L2: IF Y > 8 GOTO 6260
6350 PRINT "You trip on the slimy thing and fall!": GOTO 6370
```

If, on the other hand, the character is carrying the rope, he will be asked if he wants to use it in the climb:

```
6270 INPUT "Do you use your rope";X$: X$ = LEFT$(X$,1)
6280 IF X$ = "Y" GOTO 6300
6290 GOTO 6240
```

If for some reason the player responds "NO", everything will proceed as if the rope was not present. If he says "YES", the fall factor (X) will be given a random value of 1 to 20, instead of 1 to 10. Ignoring the slimy thing (which functions in the same way with or without the rope), the chances of successfully climbing the mountain are increased from 30 percent to 35 percent.

Table 4.23 Climb.

```
 850 IF Q = 8 AND QX$ = "CLI" GOTO 6200

6199 REM * CLIMB MOUNTAIN *
6200 INPUT "DIRECTION";Q$: Q$ = LEFT$(Q$,1)
6210 IF (Q$ = "N") OR (Q$ = "S") OR (Q$ = "E") OR (Q$ = "W") GOTO 6230
6220 GOTO 6200
6230 IF S(6) = 1 GOTO 6270
6240 X = RND(10)
6250 IF J(12) = 1 GOTO 6320
6260 IF X > 7 GOTO 720 ELSE GOTO 6360
6270 INPUT "Do you use your rope";X$: X$ = LEFT$(X$,1)
6280 IF X$ = "Y" GOTO 6300
6290 GOTO 6240
6300 X = RND(20): IF J(12) = 1 GOTO 6320
6310 GOTO 6260
6320 Y = RND(12): IF Y < 5 GOTO 6260
6330 IF S(12) = 0 GOTO 6360 ELSE PRINT "Half way up the mountain, you drop
     the slimy thing!"
6340 J(12) = 0: C(12) = L1: D(12) = L2: IF Y > 8 GOTO 6260
6350 PRINT "You trip on the slimy thing and fall!": GOTO 6370
6360 PRINT "You fall!"
6370 DG = DG - RND(DG) + 1
6380 GOTO 500
```

OPEN BOX

Table 4.24 lists the programming to add a command to open the mysteriously humming box (T(9)). The character must be carrying the box for this command to be recognized. No other object may be opened:

```
870 IF T(9)=1 AND Q$ = "OPEN BOX" GOTO 6400
```

Ordinarily, when the box is opened, it hums for a few seconds, and then snaps shut. Up to three of the junk items being carried by the explorer may vanish without a trace:

```
6400 PRINT: PRINT"  ","Hmmmm...": PRINT: GOSUB 10000
6430 FOR X = 1 TO 3: Y = RND(12): IF J(Y)=1 THEN J(Y)=0
6440 NEXT
6450 PRINT "The box snaps shut!": PRINT"  ","CLICK": PRINT
6460 GOTO 500
```

OPENing the BOX in front of a purofolee (Q = 6) has a unique effect—it causes the purofolee to vanish into thin air:

```
6410 IF Q = 6 GOTO 6470
6470 PRINT: PRINT"  ","*** POOF ***": PRINT
6480 GOSUB 10000: Q = 0: LC(L1,L2)=0: EX(L1,L2)=0
6490 PRINT "The purofolee vanishes.": PRINT: GOSUB 10000:
     GOTO 6450
```

OPENing the BOX when a kufu is present, also has a special effect—the kufu is magically transformed into a Grimph (an even nastier monster):

```
6500 PRINT"  ","*** POOF ***": PRINT: GOSUB 10000
6510 LC(L1,L2)=5: EX(L1,L2)=5: Q = 5
6520 PRINT "The kufu just turned into a Grimph!"
6530 GOTO 6450
```

OPENing the BOX has no special results in front of a Grimph, Brinchley Beast, or other obstacle.

Table 4.24 Open Box.

```
870  IF T(9)=1 AND Q$ = "OPEN BOX" GOTO 6400

6399 REM * OPEN BOX *
6400 PRINT: PRINT"  ","Hmmmm...": PRINT: GOSUB 10000
6410 IF Q = 6 GOTO 6470
```

```
6420 IF Q = 4 GOTO 6500
6430 FOR X = 1 TO 3: Y = RND(12): IF J(Y) = 1 THEN J(Y) = 0
6440 NEXT
6450 PRINT "The box snaps shut!": PRINT" ","CLICK": PRINT
6460 GOTO 500
6470 PRINT: PRINT" ","*** POOF ***": PRINT
6480 GOSUB 10000: Q = 0: LC(L1,L2) = 0: EX(L1,L2) = 0
6490 PRINT "The purofolee vanishes.": PRINT: GOSUB 10000: GOTO 6450
6500 PRINT" ","*** POOF ***": PRINT: GOSUB 10000
6510 LC(L1,L2) = 5: EX(L1,L2) = 5: Q = 5
6520 PRINT "The kufu just turned into a Grimph!"
6530 GOTO 500
```

EXPANDING "PRAY" COMMAND

When we set up the "**PRAY**" command in Chapter 3, we left space to add special results of praying when facing monsters.

If you **PRAY** in front of a Brinchley Beast and are not carrying the statue of the three-armed Martian god, not much will come of the action:

```
6550 IF Q = 3 GOTO 6600
6600 IF T(4) = 1 GOTO 6620
6610 PRINT "Nothing much seems to happen.": DG = DG + 1:
     GOTO 500
```

However, if you have the statue (T(4) = 1), the results are dramatic — the relatively harmless Brinchley Beast vanishes and is replaced by a hungry kufu:

```
6620 PRINT: PRINT" ","*** POOF ***": PRINT: GOSUB 10000
6630 PRINT "The Brinchley Beast turns into a kufu!"
6640 LC(L1,L2) = 4: EX(L1,L2) = 4: Q = 4: GOTO 6670
```

That three-armed Martian god is apparently not too fond of earthlings.

PRAYing in front a kufu is also not much of a help. The Martian gods ignore you, and the monster takes the chance to attack:

```
6560 IF Q = 4 GOTO 6650
6650 PRINT "Kufus aren't known for being pious.": PRINT:
     GOSUB 10000
6660 PRINT "It takes advantage of your inattention to attack!":
     PRINT
```

The kufu attack sequence (beginning at line 6670) was written earlier in this chapter.

Table 4.25 Pray.

```
 880  IF QX$ = "PRA" GOTO 6550

6440  NEXT: IF QX$ = "PRA" GOTO 500

6550  IF Q = 3 GOTO 6600
6560  IF Q = 4 GOTO 6650
6570  IF Q = 5 GOTO 6760
6580  IF Q = 6 GOTO 6800
6590  DG = DG + 1: PRINT: PRINT "GIMME THAT OLD TIME RELIGION!":
      PRINT: GOTO 500
6600  IF T(4) = 1 GOTO 6620
6610  PRINT "Nothing much seems to happen.": DG = DG + 1: GOTO 500
6620  PRINT: PRINT"  ","*** POOF ***": PRINT: GOSUB 10000
6630  PRINT "The Brinchley Beast turns into a kufu!"
6640  LC(L1,L2) = 4: EX(L1,L2) = 4: Q = 4; GOTO 6670
6650  PRINT "Kufus aren't known for being pious.": PRINT: GOSUB 10000
6660  PRINT "it takes advantage of your inattention to attack!": PRINT
6760  PRINT: PRINT "SORRY ---": PRINT: GOSUB 10000
6770  PRINT "All Martian deities are busy at the moment.": PRINT: GOSUB
      10000
6780  PRINT "Please call again.": PRINT: GOSUB 10000
6790  PRINT"  ","HAVE A NICE DAY!!": PRINT: PRINT: GOTO 500
6800  IF S(5) = 1 GOTO 6760 ELSE GOTO 6470
```

"**PRAY**"ing won't help much when you deal with a Grimph. The Martian gods politely brush you off:

```
6570 IF Q = 5 GOTO 6760
6760 PRINT: PRINT "SORRY —": PRINT: GOSUB 10000
6770 PRINT "All Martian deities are busy at the moment.": PRINT:
     GOSUB 10000
6780 PRINT "Please call again.": PRINT: GOSUB 10000
6790 PRINT "  ","HAVE A NICE DAY!!": PRINT: PRINT:
     GOTO 500
```

If your problem is a purofolee, **PRAY**ing may or may not help. The Martian gods are somewhat pacifistic, and if you are carrying a laser, your prayers will be ignored, and you'll get the same message for "**PRAY**"ing in front of a Grimph.

However, if you do not have the laster (S(5) = 0), the purofolee will vanish:

```
6580 IF Q = 6 GOTO 6800
6800 IF S(5) = 1 GOTO 6760 ELSE GOTO 6470
```

We can use the same purofolee vanishing routine set up for the OPEN BOX command, if we change line 6440 to read:

```
6440 NEXT: IF QX$ = "PRA" GOTO 500
```

For all other obstacles, the "**PRAY**" command functions in its normal manner, as described back in Chapter 3.

"KILL" COMMAND

Sooner or later, the explorer will find himself facing a monster that won't let him pass no matter what he does. When this happens, the player has only one choice—**KILL** it. Table 4.26 lists the programming for the "**KILL**" command. This routine is rather lengthy, but not very complex.

The "**KILL**" command is really only appropriate when facing one of the four active monsters—Brinchley Beast, kufu, Grimph, or purofolee. After all, it is not appropriate to **kill** a river.

If you use the "**KILL**" command inappropriately, the computer displays a sarcastic message, and subtracts two points from the character's current health rating (DG):

```
7900 IF (Q>2) AND (Q<7) GOTO 7930
7910 PRINT "My, but we're in a hostile mood today!"
7920 DG = DG - 2: GOTO 500
```

When you decide to **KILL** a monster, you must select a weapon:

```
7930 INPUT "Your choice of weapon";Q$
```

Ten potential weapons are recognized by the program. They are KNIFE (S(3)), GUN (S(4)), LASER (S(5)), ROPE (S(6)), METAL PIPE (S(9)), ROCK (J(3)), SHARPENED STICK (J(7)), PETRIFIED WAD OF BUBBLE GUM (J(9)), SLIMY THING (J(12)), and LARGE SWORD (T(10)). Anything else entered causes an error message to be displayed:

```
7990 PRINT "THAT DOESN'T SOUND LIKE A VERY GOOD
     WEAPON TO ME,  ";N$
7995 GOTO 500
```

For simplicity, in the following discussion we will concentrate on the KNIFE (S(3)). The other weapons are similarly programmed.

First, lines 7940 through 7985 are used to recognize the weapon name. A value is assigned to the variable W, depending on which weapon has been

selected. For the knife, W is set equal to 1. The program then jumps to line 8000:

```
7940 IF Q$ = "KNIFE" THEN W = 1: GOTO 8000
```

In lines 8000 through 8045, checks are made to make sure the explorer is carrying the specified weapon. If he is, the program jumps to an appropriate line number:

```
8000 IF W = 1 AND S(3) = 1 GOTO 8100
```

If the specified item is not being carried, an error message is displayed, and the monster (except for the purofolee) attacks the explorer. This is done in lines 8050 through 8090 in Table 4.26.

The Brinchley Beast attack routine appears at line 7780, the kufu at 6670, and the Grimph at 9100. Purofolees do not attack. Programming for the various attack routines is listed in this chapter.

Assuming the requested weapon (in this case the knife) is available, the variable X is set to a value determined by the character's attributes—specifically, aim (AX), speed (SX) and power (PX):

```
8100 X = SX + PX + (AX/2): DG = DG - 10
```

The exertion involved in using each weapon also results in a decrease of the current health rating (DG).

Each of the monsters will respond to each of the 10 weapons in different ways. To attack a Brinchley Beast with a knife, a random number from 1 to 200 is selected (Y). If this number is greater than X, the Brinchley Beast will survive the attack and bite the explorer. If X happens to be greater than Y, the Brinchley Beast will be killed.

```
8110 IF Q = 3 GOTO 8160
8160 Y = RND(200): IF Y>X GOTO 7780 ELSE GOTO 8190
8190 PRINT "GOT 'EM!": LC(L1,L2) = -Q: EX(L1,L2) = -Q: IF Q
     = 3 THEN SV = SV + 1
8198 Q = -1: GOTO 500
```

The kufu is programmed in the same way, except Y is a random number from 1 to 300. This means there is a better chance of Y being greater than X. Therefore, a kufu is harder to kill with a knife than a Brinchley Beast:

```
8120 IF Q = 4 GOTO 8170
8170 Y = RND(300): IF Y>X GOTO 6670 ELSE GOTO 8190
8192 IF Q = 4 THEN SV = SV + 3
```

The SV (monster killed score) value is incresed by a value appropriate to the monster killed. Brinchley Beasts are worth 1, purofolees 2, kufus 3, and Grimphs 4. This is in line with how tough it is to kill each of these creatures.

The Grimph is the toughest to kill. For the knife, the Grimph's Y value may be as high as 400:

```
8130 IF Q = 5 GOTO 8180
8180 Y = RND(400): IF Y>X GOTO 9100
8194 IF Q = 5 THEN SV = SV + 4
```

The purofolee is programmed basically the same as the other monsters with one minor exception. Since the purofolee does not attack, add a little personality by having it cry out. If it is killed it goes:

GIB-bleck...

If it survives your attack, it protests with an indignant:

GIBBLE!

For the knife, the purofolee's maximum Y value is the same as the Brinchley Beast--200:

```
8140 Y = RND(200): IF >X PRINT"  ", "GIBBLE!": GOTO500
8150 PRINT"  ","GIB-bleck...": GOTO 8190
8196 IF Q = 6 THEN SV = SV + 2
```

The programming continues along the same basic lines for each of the nine other weapons with changes in the X and Y values. In some cases these values are not relevant. For instance, you'll never be able to kill a Brinchley Beast with a rope.

The effects of each weapon on each monster are summarized in Tables 4.27 through 4.30. The minimum value of X assumes the character values (PX, AX, and SX) were automatically generated. If manually entered values are used, it is possible to come up with lower values of X, making the monsters tougher to kill.

"TOUCH" COMMAND

The last command we will be adding to the MARS program is "TOUCH". The programming for this command is listed in Table 4.31.

The "TOUCH" command is recognized only when a Brinchley Beast, kufu, or Grimph is present:

```
940 IF QX$ = "TOU" AND Q>2 AND Q<6 GOTO 7700
```

but the command is only appropriate for a Brinchley Beast (Q = 3). If a player attempts to **TOUCH** a kufu or Grimph, the computer will display a warning message, and 5 points will be subtracted from the current health rating (DG):

```
7700 IF Q = 3 GOTO 7730
7710 PRINT "I WOULDN'T ADVISE THAT,  ";N$
7720 DG = DG - 5: GOTO 500
```

You should recall that when a Brinchley Beast is first encountered, the variable TB is set to 0. This is the Touch Beast counter. Touching a Brinchley Beast once is good, but touching it twice or more is not beneficial.

When the **TOUCH** command is entered when a Brinchley Beast is present, the value of TB is checked:

```
7730 IF TB = 1 GOTO 7780
```

Assuming that TB is still set at 0, up to six supply items will be added to the explorer's inventory regardless of their previous location. (If the player already has the selected item, nothing will happen.)

This feature can come in handy for recovering lost items like inflatable raft (after it has been used), compass (after it has been broken by an angry Grimph), or food (after it has been eaten). Of course, the player is given no choice in which supplies the Brinchley Beast will endow on him:

```
7750 FOR X = 1 TO 6: Y = RND(12): S(Y) = 1: A(Y) = 0: B(Y) = 0:
     NEXT
```

The current health rating (DG) is also increased by up to 25 percent of its current value:

```
7760 DG = DG + (DG * .25): IF DG > DX THEN DG = DX
```

Finally, TB is set to a value of 1:

```
7770 TB = 1: GOTO 500
```

On the second attempt to touch a Brinchley Beast, TB will equal 1, and instead of bestowing gifts, the creature will bite one of 7 randomly selected portions of the explorer's anatomy. This is done in lines 7780 through 7860. This is also the Brinchley Beast attack routine called from the KILL routine previously programmed.

Table 4.26 Kill.

```
 910  IF QX$ = "KIL" GOTO 7900

7899  REM * KILL *
7900  IF (Q>2) AND (Q<7) GOTO 7930
7910  PRINT "My, but we're in a hostile mood today!"
7920  DG = DG - 2: GOTO 500
7930  INPUT "Your choice of weapon";Q$
7940  IF Q$ = "KNIFE" THEN W = 1: GOTO 8000
7945  IF Q$ = "GUN" THEN W = 2: GOTO 8000
7950  IF Q$ = "LASER" THEN W = 3: GOTO 8000
7955  QY$ = RIGHT$(Q$,4): IF QY$ = "ROPE" OR QY$ = "COIL" THEN
      W = 4: GOTO 8000
7960  IF QY$ = "PIPE" THEN W = 5: GOTO 8000
7965  IF QY$ = "ROCK" THEN W = 6: GOTO 8000
7970  IF QY$ = "TICK" THEN W = 7: GOTO 8000
7975  IF QY$ = "  WAD" OR QY$ = "  GUM" THEN W = 8: GOTO 8000
7980  IF QY$ = "HING" THEN W = 9: GOTO 8000
7985  IF QY$ = "WORD" THEN W = 10: GOTO 8000
7990  PRINT "THAT DOESN'T SOUND LIKE A VERY GOOD WEAPON TO
      ME, ": N$
7995  GOTO 500
8000  IF W = 1 AND S(3) = 1 GOTO 8100
8005  IF W = 2 AND S(4) = 1 GOTO 8200
8010  IF W = 3 AND S(5) = 1 GOTO 8300
8015  IF W = 4 AND S(6) = 1 GOTO 8400
8020  IF W = 5 AND S(9) = 1 GOTO 8500
8025  IF W = 6 AND J(3) = 1 GOTO 8600
8030  IF W = 7 AND J(7) = 1 GOTO 8700
8035  IF W = 8 AND J(9) = 1 GOTO 8800
8040  IF W = 9 AND J(12) = 1 GOTO 8900
8045  IF W = 10 AND T(10) = 1 GOTO 9000
8050  PRINT "YOU DON'T HAVE IT!!": PRINT
8060  IF Q = 3 GOTO 7780
8070  IF Q = 4 PRINT "The kufu is pleased by your error!": GOTO 6670
8080  IF Q = 5 GOTO 9100
8090  DG = DG - 3: GOTO 500
8099  REM * KNIFE *
8100  X = SX + PX + (AX/2):DG = DG - 10
8110  IF Q = 3 GOTO 8160
8120  IF Q = 4 GOTO 8170
8130  IF Q = 5 GOTO 8180
8140  Y = RND(200): IF Y > X PRINT"  ","GIBBLE!": GOTO 500
8150  PRINT"  ","GIB-bleck. . .": GOTO 8190
8160  Y = RND(200): IF Y > X GOTO 7780 ELSE GOTO 8190
```

```
8170  Y=RND(300): IF Y>X GOTO 6670 ELSE GOTO 8190
8180  Y = RND(400): IF Y>X GOTO 9100
8190  PRINT "GOT 'EM!": LC(L1,L2) = -Q: EX(L1,L2) = -Q: IF Q = 3 THEN
      SV = SV + 1
8192  IF Q = 4 THEN SV = SV + 3
8194  IF Q = 5 THEN SV = SV + 4
8196  IF Q = 6 THEN SV = SV + 2
8198  Q = -Q: GOTO 500
8199  REM * GUN *
8200  PRINT" ","* BANG! *": PRINT: DG = DG - 2
8210  X = AX + (SX/2) + (PX/5)
8220  IF Q = 3 GOTO 8260
8230  IF Q = 4 GOTO 8270
8240  IF Q = 5 GOTO 8280
8250  Y = RND(250): IF Y>X PRINT" ","GIBBLE!": GOTO 500
8255  GOTO 8150
8260  Y = RND(90): IF Y>X GOTO 7780 ELSE GOTO 8190
8270  Y = RND(370): IF Y>X GOTO 6670 ELSE GOTO 8190
8280  Y = RND(400): IF Y>X GOTO 9100 ELSE GOTO 8190
8299  REM * LASER *
8300  PRINT" ","Zzzap!!": PRINT: DG = DG - 1.5
8310  X = AX + (SX/2) + (PX/5)
8320  IF Q = 3 GOTO 8360
8330  IF Q = 4 GOTO 8370
8340  IF Q = 5 GOTO 9100
8350  Y = RND(300): IF Y>X PRINT" ","GIBBLE!": GOTO 500
8355  GOTO 8150
8360  Y = RND(230): IF Y>X GOTO 7780 ELSE GOTO 8190
8370  Y = RND(300): IF Y>X GOTO 6670 ELSE GOTO 8190
8399  REM * ROPE *
8400  DG = DG - 15
8410  IF Q = 3 GOTO 7780
8420  IF Q = 4 GOTO 6670
8430  IF Q = 5 GOTO 9100
8440  Y = RND(250): IF Y<100 GOTO 8150
8450  PRINT" ","GIBBLE!": GOTO 500
8499  REM * PIPE *
8500  DG = DG - 12: X = PX + SX + AX
8510  IF Q = 3 GOTO 8560
8520 IF Q = 4 GOTO 8570
8530  IF Q = 5 GOTO 8580
8540  Y = RND(400): IF Y>X PRINT" ","GIBBLE!": GOTO 500
8550  GOTO 8150
8560  Y = RND(350): IF Y>X GOTO 7780 ELSE GOTO 8190
8570  Y = RND(500): IF Y>X GOTO 6670 ELSE GOTO 8190
```

```
8580  Y = RND(700): IF Y>X GOTO 9100 ELSE GOTO 8190
8599  REM * ROCK *
8600  DG = DG - 12: X = PX + SX + (AX/3)
8610  IF Q = 3 GOTO 8660
8620  IF Q = 4 GOTO 6670
8630  IF Q = 5 GOTO 8680
8640  PRINT" ","GIBBLE!": GOTO 500
8660  Y = RND(300): IF Y>X GOTO 7780 ELSE GOTO 8190
8680  Y = RND(500): IF Y>X GOTO 9100 ELSE GOTO 8190
8699  REM * STICK *
8700  DG = DG - 15: X = SX + (AX/2) + (PX/2)
8710  IF Q = 3 GOTO 8760
8720  IF (Q = 4) OR (Q = 5) GOTO 8770
8730  Y = RND(350): IF Y>X PRINT" ","GIBBLE!": GOTO 500
8740  GOTO 8150
8760  Y = RND(350): IF Y>X GOTO 7780 ELSE GOTO 8190
8770  PRINT "The stick snaps into pieces.": DG = DG - 2: J(8) = 0
8780  IF Q = 4 GOTO 6670 ELSE GOTO 9100
8799  REM * GUM *
8800  DG = DG - 6: X = AX + (SX/2) + (PX/10)
8810  IF Q = 3 GOTO 7780
8820  IF Q = 4 GOTO 6670
8830  IF Q = 5 GOTO 8860
8840  Y = RND(260): IF Y>X PRINT" ","GIBBLE!": GOTO 500
8850  GOTO 8150
8860  Y = RND(200): IF Y>X GOTO 9100 ELSE GOTO 8190
8899  REM * SLIMY THING *
8900  IF Q = 3 GOTO 8950
8910  IF Q = 4 GOTO 6670
8920  IF Q = 5 GOTO 8190
8930  PRINT" ","GIBBLE!": GOTO 500
8950  PRINT "The Brinchley Beast turns into a hungry kufu!"
8960  LC(L1,L2)=4: EX(L1,L2)=4: Q = 4: GOTO 6670
8999  REM * SWORD *
9000  X = PX + (SX/2) + (AX/2): DG = DG - 5
9010  IF Q = 3 GOTO 8190
9020  IF Q = 4 GOTO 9050
9030  IF Q = 5 GOTO 9070
9040  GOTO 8150
9050  Y = RND(333): IF Y>X GOTO 6670 ELSE GOTO 8190
9070  Y = RND(250): IF Y>X GOTO 9100 ELSE GOTO 8190
```

Table 4.27 Weapons Against Brinchley Beast.

Weapon	Y Max	X Min	Percent Success	X Max	Percent Success
1—knife	200	125	62.5	250	100
2—gun	90	85	94	170	100
3—laser	230	85	37	170	74
4—rope	—	—	0	—	0
5—pipe	350	150	43	300	86
6—rock	300	117	39	233	78
7—stick	350	100	28.5	200	57
8—gum	—	—	0	—	0
9—slimy thing		Brinchley Beast turns into a kufu			
10—sword	—	—	100	—	100

Table 4.28 Weapons Against Kufu.

Weapon	Y Max	X Min	Percent Success	X Max	Percent Success
1—knife	300	125	41.5	250	83
2—gun	370	85	23	170	46
3—laser	300	85	28	170	56.5
4—rope	—	—	0	—	0
5—pipe	500	150	30	300	60
6—rock	—	—	0	—	0
7—stick	—	—	0	—	0
8—gum	—	—	0	—	0
9—slimy thing	—	—	0	—	0
10—sword	333	100	30	200	60

Table 4.29 Weapons Against Grimph.

Weapon	Y Max	X Min	Percent Success	X Max	Percent Success
1—knife	400	125	31	250	62.5
2—gun	400	85	21	170	42.5
3—laser	—	—	0	—	0
4—rope	—	—	0	—	0
5—pipe	700	150	21	300	43

6—rock	500	117	23.5	233	46.5
7—stick	—	—	0	—	0
8—gum	200	80	40	160	80
9—slimy thing	—	—	100	—	100
10—sword	250	100	40	200	80

Table 4.30 Weapons Against Purofolee.

Weapon	Y Max	X Min	Percent Success	X Max	Percent Success
1—knife	200	125	62.5	250	100
2—gun	250	85	34	170	68
3—laser	300	85	28	170	56.5
4—rope	250	100	40	100	40
5—pipe	400	150	37.5	300	75
6—rock	—	—	0	—	0
7—stick	350	100	28.5	200	57
8—gum	260	80	31	160	61.5
9—slimy thing	—	—	0	—	0
10—sword	—	—	100	—	100

Table 4.31 Touch.

```
940  IF QX$ = "TOU" AND Q>2 AND Q<6 GOTO 7700

7699 REM * TOUCH *
7700 IF Q = 3 GOTO 7730
7710 PRINT "I WOULDN'T ADVISE THAT, ":N$
7720 DG = DG - 5: GOTO 500
7730 IF TB = 1 GOTO 7780
7740 PRINT "Your hand feels rather strange..."
7750 FOR X = 1 TO 6: Y = RND(12): S(Y) = 1: A(Y) = 0: B(Y) = 0: NEXT
7760 DG = DG + (DG * .25): IF DG > DX THEN DG = DX
7770 TB = 1: GOTO 500
7780 X = RND(7): PRINT "The Brinchley Beast bites your ";
7790 IF X = 1 PRINT "hand!": DG = DG - 4
7800 IF X = 2 PRINT "arm!": DG = DG - 6
7810 IF X = 3 PRINT "foot!": DG = DG - 5
7820 IF X = 4 PRINT "toe!": DG = DG - 2.5
7830 IF X = 5 PRINT "ankle!": DG = DG - 5
7840 IF X = 6 PRINT "posterior!": DG = DG - 8
7850 IF X = 7 PRINT "kneecap!": DG = DG - 6
7860 GOTO 500
```

SUMMARY

This is now a complete adventure game program. The new routines added in this chapter are summarized in Table 4.32. Table 4.33 lists the variables added in this chapter.

The only thing now missing from the MARS game is the instructions to be written in the next chapter.

Table 4.32 Routines and Subroutines for the "Mars" Program Presented in Chapter 4.

Routines

500-620	determine obstacle present
1000-1070	move past monsters
5180-5290	CRY
5300-5680	EAT
5810-5890	DRINK at RIVER
5900-6090	FILL
6100-6190	INFLATE
6200-6380	CLIMB
6400-6530	OPEN BOX
6550-6660	PRAY
6670-6750	kufu attack
6760-6800	PRAY (continued)
7700-7770	TOUCH Brinchley Beast
7780-7860	Brinchley Beast attack
7900-9070	KILL
9100-9230	Grimph attack
9300-9330	Brinchley Beast attack
9350-9420	kufu present
9450-9460	Grimph present
9500-9570	purofolee present
9600-9680	River
9700-9810	move past River or Mountain

Subroutines

10850-10930	display dead monsters
10950-10960	Squeanly Serpent
11000-11120	Ghost
11900-12000	Ravine
12050-12160	Marsquake
12200-12270	storm

12300-12380	Funny colored sky
12400-12580	MAP
12600-12640	set up monster blocked moves

Table 4.33 Additional Variables for the "Mars" Program Added in Chapter 4.

MM	directional move allowed by monster
RV	river quality
TB	Touch Brinchley Beast counter
W	weapon used

See also Tables 3.1 and 3.18.

Chapter 5

Writing the Instructions

Once you've written an adventure game, add a set of instructions for the game so other people can play it. If it's any good at all, you'll want to share it sooner or later. Take pride in your creation.

You may put the game away for awhile and play it again a few months later. You may well have forgotten the rules yourself by this time.

An instruction routine does not take up much memory space and is a worthwhile addition to any game program. Once in awhile you may come up with a game so complex that it uses all of the memory by itself, so there is no room for instructions. This is the only circumstance when instructions should be left out. Even then, it is a good idea to write out the instructions and keep them with the tape or disc containing the program.

If it comes down to a choice between including instructions or adding special features like graphics or sound effects, opt for the instructions. Later, marketing and selling ideas for your game programs will be given. Commercial programs must *always* include clear instructions since you cannot be there to tell the user all the things you forgot to include in the instructions.

PURPOSE

The first purpose of the instruction routine is to set up the game story. For the game of "MARS" we can start out by telling the player:

> "You are on a historic mission to the planet Mars. A glorious civilization once thrived here. Your mission is to locate as many treasures as possible, return them to your space ship and blast off. But avoid excess junk weight. You will run into many monsters and dangers while you explore the mysterious Martian landscape."

Keep this introduction to the story simple and concise. Only tell the player what he needs to know. Consider the information contained in the introduction to "MARS" in chapter 2. The player now knows the setting of the game (the planet Mars), the goal (recovery of ancient treasures), and the end of game condition (blasting off the space ship). The player is also told that there will be some junk among the treasures to avoid.

The last sentence about running into monsters and dangers isn't really necessary since they are an assumed and integral part of any adventure game. But the simple reminder in the introduction helps set up the atmosphere.

What you do not tell the player is as important as what you do tell him. Don't describe the monsters. And especially don't tell how to kill them! Let the player discover these things for himself. It's more fun.

However, you will need to tell the player what kind of responses will be expected of him. If the computer asks for a command, and the player has no idea of what to do, the game can fall flat.

"MARS" is written to accept one or two word commands (or one letter commands for directional moves). The instructions should make this information clear. It's a good idea to include a few examples.

We should also emphasize that only the first and last word are recognized in "MARS". If the player enters "GET GOLD AND CUP", the computer will treat the command as if it was just "GET CUP". The gold will be ignored.

The full instruction routine for the "MARS" program is listed in Table 5.1. Notice that it consists essentially of just **PRINT** commands. An instruction routine should not be at all complex.

In writing the instructions, carefully keep track of how much information is being displayed on the screen at any time. Obviously, no information should be allowed to scroll off the top of the screen before the player has a chance to read and digest it. Delay loops could be included after each full screen of information to allow time for reading. But different people read at different rates. The instructions might go by too fast for some players, and yet be deadly dull for others.

A better approach is to include a dummy **INPUT** statement (as in lines 10110, 10200, and 10330). This allows the player to determine himself when the screen will be erased and new information displayed. Be sure to prompt the player to hit the '**ENTER**' (or '**RETURN**') key. Don't assume that he will know what to do.

Once the program is finished, including the instructions, let several friends try the game. Tell them only what is included in the instructions displayed by the program. Anything they might have trouble with should be covered in the instructions. Rewrite the subroutine if necessary.

But try not to be *too* helpful in writing the instructions. If their trouble is "How the heck do I get rid of this damn Grimph?!", that's just part of the game. Of course, if *nobody* can get past a Grimph, you may have made things too tough. Don't conclude that you just happen to have stupid friends and refuse to change anything. They probably represent other players well.

Table 5.1 The Instructions Subroutine for the "Mars" Program.

```
10010 PRINT: PRINT "YOUR MISSION,  ";N$; "  IS TO EXPLORE THE"
10020 PRINT "MYSTERIOUS PLANET MARS!": PRINT
10030 PRINT "An ancient civilization once thrived here. You must find as"
10040 PRINT "many of the treasured relics from this long-dead culture as"
10050 PRINT "you can, return them to your rocket ship, and blast off for"
10060 PRINT "good old Earth.": PRINT
10070 PRINT "But watch out—some items you will find may be worthless"
10080 PRINT "junk that will have a negative effect on your score."
10090 PRINT "Some of the junk might, however, come in handy during your"
10100 PRINT "exploration of the Red Planet.": PRINT
10110 INPUT "Please press 'ENTER' for more  ";Q$
10120 CLS: PRINT: PRINT "You will encounter a number of Martian beasts and"
10130 PRINT "monsters during your travels. Different types of creatures"
10140 PRINT "must be handled in different ways. The successful methods may"
10150 PRINT "not always be obvious. The bizarre natural forces of Mars"
10160 PRINT "will also tend to present obstacles.": PRINT: PRINT
10170 PRINT "You may define the characteristics of your explorer by setting"
10180 PRINT "a value from 1 to 100% for each quality (such as strength,  "
10190 PRINT "speed, etc.), or you can let the computer automatically define"
10200 PRINT "your character for you.": PRINT: INPUT "Please press 'ENTER' for
      more  ";Q$
10210 CLS: PRINT: PRINT "During the game, each command may be one or two
      words, such as;"
10220 PRINT "LOOK", "DROP ROCK", "EAT FOOD", "WAIT"
10230 PRINT "Or a single letter directional move command;"
10240 PRINT "(N = North, S = South, E = East, and W = West)"
10250 PRINT "Commands consisting of more than two words, such as;"
10260 PRINT "PUT KUFU IN POCKET", "are not valid."
10270 PRINT: PRINT "If you have no idea of what to do, you may use the special"
10280 PRINT "command HELP for a list of commnds used in the game."
10290 PRINT "NOT ALL COMMANDS WILL BE ACCEPTABLE UNDER ALL
      CIRCUMSTANCES!"
10300 PRINT "Also, what may be helpful at one time, may be of no particular"
10310 PRINT "effect at another, and actually harmful at a third time."
10320 PRINT "For instance, EATing in front of a Grimph is not recommended."
10330 PRINT: INPUT "Please press 'ENTER' to define your character  ";Q$
10340 RETURN
```

In conclusion, make the instructions clear enough so it is possible to win the game, but not so clear as to destroy the point of the game. If everybody wins, or if everybody loses, your program has problems.

Chapter 6

The Complete "MARS" Program

We have now completed the programming for the game of "MARS". It is rather lengthy and will not fit into a 16K computer without considerable editing. However, a 32K or 48K machine will have plenty of memory space.

Table 6.1 shows a print out for the complete "MARS" program.

In Chapters 8-10, three additional game programs will be presented and discussed. These programs will not be analyzed in as much depth as "MARS" was in the previous chapters. But they offer a good over-view of the various techniques and methods of writing computerized adventure games.

Have fun!

Table 6.1 Complete "Mars" Program.

```
  1  REM * MARS * Delton T. Horn *
  5  CLEAR 100: DIMA(12): DIM B(12): DIM C(12): DIM D(12): DIM E(12):
     DIM F(12)
 10  DIM S(12): DIM J(12): DIM T(12): DIM LC(10,10): DIM EX(10,10)
 20  CLS: PRINT: PRINT"  ","MARS": PRINT: INPUT"YOUR NAME";N$
 30  PRINT: INPUT"WILL YOU NEED INSTRUCTIONS";Q$:
     Q$ = LEFT$(Q$,1)
 40  FOR X = 1 TO 10: FOR Y = 1 TO 10: EX(X,Y) = 0
 50  PRINT"  *  ";: Z = RND(17): IF Z > 12 THEN Z = 0
 60  LC(X,Y) = Z: NEXT: NEXT
 70  PRINT"  PLEASE BE PATIENT,  ";N$
 80  R1 = RND(10): R2 = RND(10): L1 = R1: L2 = R2
 90  EX(R1,R2) = 20: LC(R1,R2) = 20: PRINT: PRINT"I'M BUILDING AN
     ENTIRE PLANET HERE!": PRINT
100  FOR X = 1 TO 12: A(X) = R1: B(X) = R2
110  Y = RND(10): Z = RND(10): IF Y = R1 AND Z = R2 GOTO 110
```

```
120 C(X) = Y: D(X) = Z
130 Y = RND(10): Z = RND(10): IF Y = R1 AND Z = R2 GOTO 130
140 E(X) = Y: F(X) = Z
150 S(X) = 0: J(X) = 0: T(X) = 0
160 NEXT: GOSUB 10000: INPUT"Please press 'ENTER'  ";Q$
170 IF Q$ = "Y" GOSUB 10010
180 INPUT "ENTER 1 FOR AUTOMATIC CHARACTER OR 2 TO CREATE
    YOUR OWN";X
190 IF X = 1 GOTO 210
200 IF X = 2 GOTO 230 ELSE GOTO 180
210 AX = RND(50)+50: DX = RND(100)+100: SX = RND(50)+50:
    PX = RND(50)+50
220 GOTO 300
230 INPUT "HEALTH";DX: IF DX<1 OR DX>100 GOTO 230
240 DX = DX*2: INPUT"SPEED";SX: IF SX < 1 OR SX > 100 GOTO 240
250 INPUT"POWER";PX: IF PX < 1 OR PX > 100 GOTO 250
260 INPUT"AIM";AX: IF AX < 1 OR AX > 100 GOTO 260
300 CLS: PRINT: PRINT"  ",N$: PRINT
310 PRINT"HEALTH",DX/2;"%"
320 PRINT"SPEED",SX;"%"
330 PRINT"POWER",PX;"%"
340 PRINT"AIM",AX;"%"
370 DG = DX
380 INPUT"Please press 'ENTER' to play  ";Q$: CLS: PRINT: PRINT
399 REM * LOCATION DISPLAY *
400 Q = LC(L1,L2): PRINT: PRINT"  ","Your current coordinates are  ";
405 GOSUB 12600
410 IF S(11) = 0 PRINT"?:?"ELSE PRINTL1;":";L2
420 IF L1 = R1 AND L2 = R2 GOTO 430
425 IF S(12) = 0 GOTO 5000
430 IF L1 = R1 AND L2 = R2 PRINT"You are safely aboard your rocket ship.":
    DG = DG+3
440 Y = 0: FOR X = 1 TO 12: IF A(X) = L1 AND B(X) = L2 GOSUB 10450
450 IF C(X) = L1 AND D(X) = L2 GOSUB 10570
460 IF E(X) = L1 AND F(X) = L2 GOSUB 10700
470 IF Y > 8 THEN Y = 0: INPUT"Please press 'ENTER'  ";Q$
480 NEXT X
499 REM * CHECK FOR MONSTERS *
500 EX(L1,L2) = Q: IF (Q<0) AND (Q> -7)GOSUB 10850
510 IF Q = 1 GOSUB 10950
520 IF Q = 2 GOSUB 11000
530 IF Q = 3 GOTO 9300
540 IF Q = 4 GOTO 9350
550 IF Q = 5 GOTO 9450
560 IF Q = 6 GOTO 9500
```

```
570  IF Q = 7 GOTO 9600
580  IF Q = 8 PRINT"You are at the foot of a tall, craggy mountain"
590  IF Q = 9 GOSUB 11900
600  IF Q = 10 GOSUB 12050
610  IF Q = 11 GOSUB 12200
620  IF Q = 12 GOSUB 12300
649  REM * HEALTH CHECK *
650  IF DG < 1 GOTO 5070
660  IF DG < 40 PRINT"You're not looking too well, pal."
699  REM * MAIN COMMAND *
700  Q$ = "": DG = DG - 1: PRINT: PRINT"YOUR COMMAND,  ";N$
710  INPUT Q$
715  IF(Q > 2) AND (Q < 7) GOTO 1000
717  IF(Q = 7 OR Q = 8) AND (Q$ = "N" OR Q$ = "S" OR Q$ = "E" OR
     Q$ = "W") GOTO 9700
720  IF Q$ = "N" GOTO 5100
730  IF Q$ = "S" GOTO 5120
740  IF Q$ = "E" GOTO 5140
750  IF Q$ = "W" GOTO 5160
760  QX$ = LEFT$(Q$,3): QY$ = RIGHT$(Q$,3)
770  IF QX$ = "SCO" GOSUB 10400: PRINT"YOUR SCORE SO FAR IS  ";SC:
     GOTO 700
780  IF QX$ = "DIA" GOSUB 11150: GOTO 700
790  IF QX$ = "CRY" GOTO 5180
800  IF QX$ = "EAT" GOTO 5300
810  IF QX$ = "DRI" GOTO 5700
820  IF Q = 7 AND QX$ = "FIL" GOTO 5900
830  IF S(7) = 1 AND QX$ = "INF" GOTO 6100
840  IF QX$ = "INV" GOSUB 11300: GOTO 700
850  IF Q = 8 AND QX$ = "CLI" GOTO 6200
860  IF QX$ = "LOO" GOTO 400
870  IF T(9) = 1 AND Q$ = "OPEN BOX" GOTO 6400
880  IF QX$ = "PRA" GOTO 6550
890  IF QX$ = "GET" GOTO 6810
900  IF QX$ = "DRO" GOTO 6910
910  IF QX$ = "KIL" GOTO 7900
920  IF QX$ = "HEL" GOSUB 11800: GOTO 700
930  IF QX$ = "WAI" GOTO 9820
940  IF QX$ = "TOU" AND Q > 2 AND Q < 6 GOTO 7700
950  IF QX$ = "MAP" GOSUB 12400: GOTO 700
960  QX$ = "BLA" GOTO 9960
970  GOSUB 11700
980  DG = DG - 1: GOTO 500
999  REM * MONSTER BLOCK MOVE *
1000 IF Q$ = "N" OR Q$ = "S" OR Q$ = "W" OR Q$ = "E" GOTO 1010
```

```
1005 GOTO 760
1010 IF Q$ = "N" AND MM = 1 GOTO 5100
1015 IF Q$ = "S" AND MM = 2 GOTO 5120
1020 IF Q$ = "E" AND MM = 3 GOTO 5140
1025 IF Q$ = "W" AND MM = 4 GOTO 5160
1030 IF Q = 5 GOTO 9100
1040 IF Q = 3 PRINT"THE BRINCHLEY BEAST BLOCKS YOUR PATH."
1050 IF Q = 4 PRINT"THE KUFU WILL NOT LET YOU GO THAT WAY.":
     DG = DG - 2
1060 IF Q = 6 PRINT"The durn purofolee is in your way!"
1070 GOTO 650
4999 STOP
5000 PRINT"You are outside without your spacesuit!": PRINT
5010 FOR X = 1 TO 4: Y = RND(5): Z = RND(75)+25: FOR ZZ = 1 TO Z:
     NEXT ZZ
5020 IF Y = 1 PRINT"* gasp *",
5030 IF Y = 2 PRINT"* choke *",
5040 IF Y = 3 PRINT"* pant-pant *",
5050 IF Y = 4 PRINT"* wheeze *",
5060 NEXT X: PRINT
5070 GOSUB 10000: PRINT: PRINT"You are deceased.": PRINT
5080 PRINT"Your final score was  ";
5090 GOSUB 10400: PRINTSC: END
5099 REM * MOVES * NORTH *
5100 K = 0: L3 = L1: L4 = L2: L1 = L1 - 1: IF L1 < 1 THEN L1 = 10
5110 GOTO 400
5119 REM * SOUTH *
5120 K = 0: L3 = L1: L4 = L2: L1 = L1 + 1: IF L1 > 10 THEN L1 = 1
5130 GOTO 400
5139 REM * EAST *
5140 K = 0: L3 = L1: L4 = L2: L2 = L2 - 1: IF L2 < 1 THEN L2 = 10
5150 GOTO 400
5159 REM * WEST *
5160 K = 0: L3 = L1: L4 = L2: L2 = L2 + 1: IF L2 > 10 THEN L2 = 1
5170 GOTO 400
5179 REM * CRY *
5180 IF Q = 4 GOTO 5220
5190 IF Q = 6 GOTO 5240
5200 PRINT"Why? Are you upset for some reason?"
5210 GOTO 500
5220 PRINT"Kufus aren't noted for being soft-hearted."
5230 GOTO 500
5240 X = RND(10): IF X > 5 GOTO 5270
5250 PRINT"Annoyed by the noise you're making, the purofolee knocks you":
     PRINT"down and jumps on your stomach 17 times!"
```

```
5260 DG = DG - 17: GOTO 500
5270 PRINT"Frightened by the noise you're making, the purofolee hops off."
5280 X = L1+RND(5) - 3: Y = L2+RND(5) - 3: IF X = L1 AND Y = L2
     GOTO 5280
5285 IF X = R1 AND Y = R2 GOTO 5280
5290 LC(L1,L2) = 0: EX(L1,L2) = 0: Q = 0: LC(X,Y) = 6: GOTO 500
5299 REM * EAT *
5300 IF Q > 3 AND Q < 7 GOTO 5600
5310 IF QY$ = "OOD" GOTO 5420
5320 IF QY$ = "UFU" GOTO 5480
5330 IF QY$ = "LEE" GOTO 5540
5340 X = LEN(Q$): IF X < 5 GOTO 5590
5350 X = RND(6): DG = DG - .5: IF X = 1 PRINT"Eat WHAT???"
5360 IF X = 2 PRINT"-er- No, thank you..."
5370 IF X = 3 PRINT"ARE YOU NUTS?"
5380 IF X = 4 PRINT"That hungry, I'm not!"
5390 IF X = 5 PRINT"YUK!"
5400 IF X = 6 PRINT"Has anyone ever told you that you have weird tastes?"
5410 GOTO 500
5420 IF S(1) = 1 GOTO 5450
5430 PRINT"YOU DON'T HAVE ANY!"
5440 DG = DG - 1: GOTO 500
5450 PRINT"  ","burp": PRINT
5460 DG = DG+(DX*.25): IF DG > DX THEN DG = DX
5470 S(1) = 0: GOTO 500
5480 IF Q = -4 GOTO 5500
5490 PRINT"First you have to go out and kill one,  ";N$: DG = DG - 1:
     GOTO 500
5500 PRINT"You have to hold your nose to get close enough,"
5510 DG = DG+RND(15) - 10: IF DG > DX THEN DG = DX
5520 PRINT"but you somehow manage to get it down.": LC(L1,L2) = 0
5530 EX(L1,L2) = 0: Q = 0: GOTO 500
5540 IF Q = -6 GOTO 5560
5550 GOTO 5490
5560 PRINT"  ","YUMMY!": PRINT: DG = DG + 50
5570 IF DG > DX THEN DG = DX
5580 LC(L1,L2) = 0: EX(L1,L2) = 0: Q = 0: GOTO 500
5590 PRINT"Eat what?": DG = DG - .25: GOTO 500
5600 IF Q = 4 GOTO 5640
5610 IF Q = 5 GOTO 5670
5620 IF S(1) = 0 PRINT"There is nothing here to eat.": DG = DG - 1: GOTO
     500
5630 PRINT"When you take your food out, the durn purofolee snatches it"
5635 PRINT"before you can take a single bite!": PRINT: S(1) = 0: DG = DG - 1:
     GOTO 500
```

```
5640 PRINT"This is no time to think of your stomach,  ";N$;"!"
5650 GOSUB 10000: PRINT"Unless you want to see the inside of a kufu's
     stomach!"
5660 DG = DG - 2: GOTO 500
5670 PRINT"The grimph thinks that's a jim dandy idea!": PRINT: GOSUB
     10000
5680 PRINT"It eats you!!": PRINT: GOTO 5070
5699 REM * DRINK *
5700 IF Q = 7 GOTO 5810
5710 IF S(2) = 1 GOTO 5740
5720 IF S(2) = 2 GOTO 5800
5730 PRINT"THERE IS NOTHING HERE TO DRINK,  ";N$: DG = DG - .5:
     GOTO 500
5740 PRINT"  ",: FOR X = 1 TO 3: FOR Y = 1 TO 85: NEXT Y
5750 PRINT"* glug *  ",: NEXT X: PRINT
5760 GOSUB 10000
5770 PRINT"  ","AHHHH!": DG = DG+(DX*.15)
5780 IF DG > DX THEN DG = DX
5790 S(2) = 2: GOTO 500
5800 PRINT"Your water bottle is empty.": DG = DG - .4: GOTO 500
5810 PRINT"  ","slurp!!!": PRINT: GOSUB 10000
5820 X = RND(10): IF X > 7 GOTO 5860
5830 IF X < 4 GOTO 5870
5840 DG = DG+DX*.15: IF DG > DX THEN DG = DX
5850 GOTO 500
5860 PRINT"It sure tastes good!": DG = DG+DX*.1: GOTO 5840
5870 PRINT"This water makes you quite ill.": DG = DG*.75
5880 PRINT"But you don't die  ";: GOSUB 10000
5890 PRINT"— at least, not just yet.": GOTO 500
5899 REM * FILL *
5900 X = LEN(Q$): IF X > 6 GOTO 5920
5910 PRINT"Fill what?": DG = DG - .5: GOTO 500
5920 IF QY$ = "TLE" GOTO 5990
5930 IF QY$ = "URN" GOTO 6040
5940 IF QY$ = "BOX" GOTO 6060
5950 IF QY$ = "UIT" PRINT"You drown!": GOTO 5070
5960 IF QY$ = "OWL" GOTO 6080
5970 IF QY$ = "CUP" GOTO 6090
5980 PRINT"THAT WON'T HOLD WATER,  ";N$: DG = DG - 1: GOTO 500
5990 IF S(2) = 2 GOTO 6030
6000 IF S(2) = 1 GOTO 6020
6010 PRINT"YOU DON'T HAVE IT,  ";N$;"!": DG = DG - 1.25: GOTO 500
6020 PRINT"It's already full": DG = DG - 1: GOTO 500
6030 PRINT"OK. The bottle is now full.": S(2) = 1: GOTO 500
6040 IF J(8) = 0 GOTO 6010 ELSE PRINT"The urn";
```

```
6050 PRINT"  leaks.": DG = DG - .75: GOTO 500
6060 IF T(9) = 0 GOTO 6010
6070 PRINT"The box dissolves.": T(9) = 0: GOTO 500
6080 IF T(1) = 0 GOTO 6010 ELSE PRINT"The copper bowl";: GOTO 6050
6090 IF T(5) = 0 GOTO 6010 ELSE PRINT"The silver cup";: GOTO 6050
6099 REM * INFLATE RAFT *
6100 PRINT"  ","Phfffft!": PRINT
6110 GOSUB 10000: IF Q = 7 GOTO 6130
6120 PRINT"The raft springs a leak!": PRINT: S(7) = 0: GOTO 500
6130 DG = DG - (RND(50)/10)
6140 PRINT"You cannot control the raft on these raging currents"
6150 PRINT: GOSUB 10000
6160 PRINT"You make it to a shore, but you lose your raft!"
6170 X = L1 - 2 + RND(3): Y = L2 - 2 + RND(3): IF X = L1 AND Y = L2
     GOTO 6170
6172 IF X < 1 THEN X = 10
6174 IF X > 10 THEN X = 1
6176 IF Y < 1 THEN Y = 10
6178 IF Y > 10 THEN Y = 1
6180 S(7) = 0: L3 = L1: L4 = L2: L1 = X: L2 = Y: INPUT"Please press
     'ENTER'  ";Q$
6190 GOTO 400
6199 REM * CLIMB MOUNTAIN *
6200 INPUT"DIRECTION";Q$: Q$ = LEFT$(Q$,1)
6210 IF (Q$ = "N") OR (Q$ = "S") OR (Q$ = "E") OR (Q$ = "W") GOTO 6230
6220 GOTO 6200
6230 IF S(6) = 1 GOTO 6270
6240 X = RND(10)
6250 IF J(12) = 1 GOTO 6320
6260 IF X > 7 GOTO 720 ELSE GOTO 6360
6270 INPUT"Do you use your rope";X$: X$ = LEFT$(X$,1)
6280 IF X$ = "Y" GOTO 6300
6290 GOTO 6240
6300 X = RND(20): IF J(12) = 1 GOTO 6320
6310 GOTO 6260
6320 Y = RND(12): IF Y < 5 GOTO 6260
6330 IF S(12) = 0 GOTO 6360 ELSE PRINT"Half way up the mountain, you drop
     the slimy thing!"
6340 J(12) = 0: C(12) = L1: D(12) = L2: IF Y > 8 GOTO 6260
6350 PRINT"You trip on the slimy thing and fall!": GOTO 6370
6360 PRINT"You fall!"
6370 DG = DG - RND(DG) + 1
6380 GOTO 500
6399 REM * OPEN BOX *
6400 PRINT: PRINT"  ","Hmmmm. . .": PRINT: GOSUB 10000
```

```
6410 IF Q = 6 GOTO 6470
6420 IF Q = 4 GOTO 6500
6430 FOR X = 1 TO 3: Y = RND(12): IF J(Y) = 1 THEN J(Y) = 0
6440 NEXT: IF QX$ = "PRA" GOTO 500
6450 PRINT"The box xnaps shut!": PRINT"  ","CLICK": PRINT
6460 GOTO 500
6470 PRINT: PRINT"  ","*** POOF ***": PRINT
6480 GOSUB 10000: Q = 0: LC(L1,L2) = 0: EX(L1,L2) = 0
6490 PRINT"The purofolee vanishes.": PRINT: GOSUB 10000: GOTO 6450
6500 PRINT"  ","*** POOF ***": PRINT: GOSUB 10000
6510 LC(L1,L2) = 5: EX(L1,L2) = 5: Q = 5
6520 PRINT"The kufu just turned into a Grimph!"
6530 GOTO 500
6550 IF Q = 3 GOTO 6600
6560 Q = 4 GOTO 6650
6570 IF Q = 5 GOTO 6760
6580 IF Q = 6 GOTO 6800
6590 DG = DG + 1: PRINT: PRINT"GIMME THAT OLD TIME RELIGION!":
     PRINT: GOTO 500
6600 IF T(4) = 1 GOTO 6620
6610 PRINT"Nothing much seems to happen.": DG = DG + 1: GOTO 500
6620 PRINT: PRINT"  ","*** POOF ***": PRINT: GOSUB 10000
6630 PRINT"The Brinchley Beast turns into a kufu!"
6640 LC(L1,L2) = 4: EX(L1,L2) = 4: Q = 4: GOTO 6670
6650 PRINT"Kufus aren't known for being pious": PRINT: GOSUB 10000
6660 PRINT"It takes advantage of our inattention to attack!": PRINT
6670 PRINT"It bites your  ";: X = RND(7)
6680 IF X = 1 PRINT"arm!": DG = DG – 5
6690 IF X = 2 PRINT"leg!": DG = DG – 7
6700 IF X = 3 PRINT"stomach!": DG = DG – 50
6710 IF X = 4 PRINT"neck!": DG = DG – 75
6720 IF X = 5 PRINT"air hose!": GOTO 5070
6730 IF X = 6 PRINT"nose!": DG = DG – 20
6740 IF X = 7 PRINT"posterior!": DG = DG – 30
6750 GOTO 650
6760 PRINT: PRINT"SORRY –  ": PRINT: GOSUB 10000
6770 PRINT"All Martian deities are busy at the moment.": PRINT: GOSUB
     10000
6780 PRINT"Please call again.": PRINT: GOSUB 10000
6790 PRINT"  ","HAVE A NICE DAY!!": PRINT: PRINT: GOTO 500
6800 IF S(5) = 1 GOTO 6760 ELSE GOTO 6470
6809 REM * GET/DROP *
6810 IF Q$ = "GET" PRINT"GET WHAT?": GOTO 500
6815 IF Q$ = "GET ALL" GOTO 9900
6820 IF QY$ = "OST" GOTO 6870
```

```
6830  IF Q$ = "GET SICK" GOTO 6880
6840  IF QY$ = "ENT" OR QY$ = "KED" GOTO 6890
6850  IF QY$ = "OWN" GOTO 6900
6860  G = 1: Y = 0: FOR X = 1 TO 12: Y = Y + S(X) + J(X) + T(X): NEXT: IF
      Y > 17 GOTO 9850 ELSE GOTO 693O
6870  PRINT"I think you already are. . .": GOTO 500
6880  PRINT"That's disgusting!": GOTO 500
6890  PRINT"SAME TO YOU,  ";N$;"!": GOTO 500
6900  PRINT"This is no time to boogie!": GOTO 500
6910  IF QY$ = "EAD" GOTO 6890
6920  G = 2
6930  IF QY$ = "OOD" U = 1: GOTO 7500
6940  IF QY$ = "TLE" OR QY$ = "TER" U = 2: GOTO 7500
6950  IF QY$ = "IFE" U = 3: GOTO 7500
6960  IF QY$ = "GUN" U = 4: GOTO 7500
6970  IF QY$ = "SER" U = 5: GOTO 7500
6980  IF QY$ = "OPE" OR QY$ = "OIL" U = 6: GOTO 7500
6990  IF QY$ = "AFT" U = 7: GOTO 7500
7000  IF QY$ = "GHT" U = 8: GOTO 7500
7010  IF QY$ = "IPE" U = 9: GOTO 7500
7020  IF RIGHT$(Q$,4) = "INES" U = 10: GOTO 7500
7030  IF QY$ = "ASS" U = 11: GOTO 7500
7040  IF QY$ = "UIT" AND G = 2 GOTO 5000
7050  IF QY$ = "UIT" U = 12: GOTO 7500
7060  IF QY$ = "HOE" U = 1: GOTO 7600
7070  IF RIGHT$(Q$,4) = "RING" U = 2: GOTO 7600
7080  IF QY$ = "OCK" U = 3: GOTO 7600
7090  IF QY$ = "RTS" U = 4: GOTO 7600
7100  IF QY$ = "IRT" OR QY$ = "LOT" U = 5: GOTO 7600
7110  IF QY$ = "ONE" U = 6: GOTO 7600
7120  IF QY$ = "ICK" U = 7: GOTO 7600
7130  IF QY$ = "URN" U = 8: GOTO 7600
7140  IF QY$ = "WAD" OR QY$ = "GUM" U = 9: GOTO 7600
7150  IF QY$ = "WER" U = 10: GOTO 7600
7160  IF QY$ = "FLY" U = 11: GOTO 7600
7170  IF RIGHT$(Q$,4) = "HING" U = 12: GOTO 7600
7180  IF QY$ = "PER" OR QY$ = "OWL" U = 1: GOTO 7650
7190  IF QY$ = "INS" U = 2: GOTO 7650
7200  IF QY$ = "ULE" U = 3: GOTO 7650
7210  IF QY$ = "TUE" OR QY$ = "GOD" U = 4: GOTO 7650
7220  IF QY$ = "VER" OR QY$ = "CUP" U = 5: GOTO 7650
7320  IF QY$ = "ORB" U = 6: GOTO 7650
7240  IF QY$ = "OLL" U = 7: GOTO 7650
7250  IF RIGHT$(Q$,4) = "ONES" U = 8: GOTO 7650
7260  IF QY$ = "BOX" U = 9: GOTO 7650
```

```
7270  IF QY$ = "ORD" U = 10: GOTO 7650
7280  IF QY$ = "ULL" U = 11: GOTO 7650
7290  IF QY$ = "NTS" U = 12: GOTO 7650
7300  X = LEN(Q$): IF X < 6 GOTO 7350
7310  IF G = 1 THEN X = X - 4 ELSE X = X - 5
7320  QY$ = RIGHT$(Q$,X)
7330  PRINT"SORRY,  ";N$;", BUT I DON'T SEE ANY  ";QY$;"  HERE."
7350  PRINT"Please try to keep your commands rational in the future."
7360  PRINT: DG = DG - 3: GOTO 500
7500  IF G = 2 GOTO 7550
7510  IF S(U) > 0 GOTO 7540
7515  IF A(U) = L1 AND B(U) = L2 GOTO 7520 ELSE GOTO 7590
7520  S(U) = 1: A(U) = 0: B(U) = 0: PRINT"OK": GOTO 500
7540  PRINT"You already have it!": DG = DG - 1: GOTO 500
7550  IF S(U) < 1 GOTO 7580
7560  S(U) = 0: A(U) = L1: B(U) = L2: PRINT"OK": GOTO 500
7580  PRINT"You don't have it!": DG = DG - 1: GOTO 500
7590  PRINT"IT'S NOT HERE,  ";N$: DG = DG - 1: GOTO 500
7600  IF G = 2 GOTO 7630
7610  IF J(U) > 0 GOTO 7540
7615  IF C(U) = L1 AND D(U) = L2 GOTO 7620 ELSE GOTO 7590
7620  J(U) = 1: C(U) = 0: D(U) = 0: PRINT"OK": GOTO 500
7630  IF J(U) < 1 GOTO 7580
7640  J(U) = 0: C(U) = L1: D(U) = L2: PRINT"OK": GOTO 500
7650  IF G = 2 GOTO 7680
7660  IF T(U) > 0 GOTO 7540
7665  IF E(U) = L1 AND F(U) = L2 GOTO 7670 ELSE GOTO 7590
7670  T(U) = 1: E(U) = 0: F(U) = 0: PRINT"OK": GOTO 500
7680  IF T(U) < 1 GOTO 7580
7690  T(U) = 0: E(U) = L1: F(U) = L2: PRINT"OK": GOTO 500
7699  REM * TOUCH *
7700  IF Q = 3 GOTO 7730
7710  PRINT"I WOULDN'T ADVISE THAT,  "N$
7720  DG = DG - 5: GOTO 500
7730  IF TB = 1 GOTO 7780
7740  PRINT"Your hand feels rather strange . . ."
7750  FOR X = 1 TO 6: Y = RND(12): S(Y) = 1: A(Y) = 0: B(Y) = 0: NEXT
7760  DG = DG + (DG*.25): IF DG > DX THEN DG = DX
7770  TB = 1: GOTO 500
7780  X = RND(7): PRINT"The Brinchley Beast bites your  ";
7790  IF X = 1 PRINT"hand!": DG = DG - 4
7800  IF X = 2 PRINT"arm!": DG = DG - 6
7810  IF X = 3 PRINT"foot!": DG = DG - 5
7820  IF X = 4 PRINT"toe!": DG = DG - 2.5
7830  IF X = 5 PRINT"ankle!": DG = DG - 5
```

```
7840 IF X = 6 PRINT"posterior!": DG = DG - 8
7850 IF X = 7 PRINT"kneecap!": DG = DG - 6
7860 GOTO 500
7899 REM * KILL *
7900 IF(Q > 2) AND (Q < 7) GOTO 7930
7910 PRINT"My, but we're in a hostile mood today!"
7920 DG = DG - 2: GOTO 500
7930 INPUT"Your choice of weapon";Q$
7940 IF Q$ = "KNIFE" THEN W = 1: GOTO 8000
7945 IF Q$ = "GUN" THEN W = 2: GOTO 8000
7950 IF Q$ = "LASER" THEN W = 3: GOTO 8000
7955 QY$ = RIGHT$(Q$,4): IF QY$ = "ROPE" OR QY$ = "COIL" THEN
     W = 4: GOTO 8000
7960 IF QY$ = "PIPE" THEN W = 5: GOTO 8000
7965 IF QY$ = "ROCK" THEN W = 6: GOTO 8000
7970 IF QY$ = "TICK" THEN W = 7: GOTO 8000
7975 IF QY$ = "  WAD" OR QY$ = "  GUM" THEN W = 8: GOTO 8000
7980 IF QY$ = "HING" THEN W = 9: GOTO 8000
7985 IF QY$ = "WORD" THEN W = 10: GOTO 8000
7990 PRINT"THAT DOESN'T SOUND LIKE A VERY GOOD WEAPON TO
     ME,  ";N$
7995 GOTO 500
8000 IF W = 1 AND S(3) = 1 GOTO 8100
8005 IF W = 2 AND S(4) = 1 GOTO 8200
8010 IF W = 3 AND S(5) = 1 GOTO 8300
8015 IF W = 4 AND S(6) = 1 GOTO 8400
8020 IF W = 5 AND S(9) = 1 GOTO 8500
8025 IF W = 6 AND J(3) = 1 GOTO 8600
8030 IF W = 7 AND J(7) = 1 GOTO 8700
8035 IF W = 8 AND J(9) = 1 GOTO 8800
8040 IF W = 9 AND J(12) = 1 GOTO 8900
8045 IF W = 10 AND T(10) = 1 GOTO 9000
8050 PRINT"YOU DON'T HAVE IT!!": PRINT
8060 IF Q = 3 GOTO 7780
8070 IF Q = 4 PRINT"The kufu is pleased by your error!": GOTO 6670
8080 IF Q = 5 GOTO 9100
8090 DG = DG - 3: GOTO 500
8099 REM * KNIFE *
8100 X = SX + PX + (AX/2): DG = DG - 10
8110 IF Q = 3 GOTO 8160
8120 IF Q = 4 GOTO 8170
8130 IF Q = 5 GOT 8180
8140 Y = RND(200): IF Y > X PRINT"  ","GIBBLE!": GOTO 500
8150 PRINT"  ","GIB-bleck...": GOTO 8190
8160 Y = RND(200): IF Y > X GOTO 7780 ELSE GOTO 8190
```

```
8170  Y = RND(300): IF Y > X GOTO 6670 ELSE GOTO 8190
8180  Y = RND(400): IF Y > X GOTO 9100
8190  PRINT"GOT 'EM!": LC(L1,L2) = -Q: EX(L1,L2) = -Q: IF Q = 3 THEN
      SV = SV + 1
8192  IF Q = 4 THEN SV = SV + 3
8194  IF Q = 5 THEN SV = SV + 4
8196  IF Q = 6 THEN SV = SV + 2
8198  Q = -Q: GOTO 500
8199  REM * GUN *
8200  PRINT"  ","* BANG!": PRINT: DG = DG - 2
8210  X = AX + (SX/2) + (PX/5)
8220  IF Q = 3 GOTO 8260
8230  IF Q = 4 GOTO 8270
8240  IF Q = 5 GOTO 8280
8250  Y = RND(250): IF Y > X PRINT"  ","GIBBLE!": GOTO 500
8255  GOTO 8150
8260  Y = RND(90): IF Y > X GOTO 7780 ELSE GOTO 8190
8270  Y = RND(370): IF Y > X GOTO 6670 ELSE GOTO 8190
8280  Y = RND(400): IF Y > X GOTO 9100 ELSE GOTO 8190
8299  REM * LASER *
8300  PRINT"  ","Zzzap!!": PRINT: DG = DG - 1.5
8310  X = AX + (SX/2) + (PX/5)
8320  IF Q = 3 GOTO 8360
8330  IF Q = 4 GOTO 8370
8340  IF Q = 5 GOTO 9100
8350  Y = RND(300): IF Y > X PRINT"  ","GIBBLE!": GOTO 500
8355  GOTO 8150
8360  Y = RND(230): IF Y > X GOTO 7780 ELSE GOTO 8190
8370  Y = RND(300): IF Y > X GOTO 6670 ELSE GOTO 8190
8399  REM * ROPE *
8400  DG = DG - 15
8410  IF Q = 3 GOTO 7780
8420  IF Q = 4 GOTO 6670
8430  IF Q = 5 GOTO 9100
8440  Y = RND(250): IF Y < 100 GOTO 8150
8450  PRINT"  ","GIBBLE!": GOTO 500
8499  REM * PIPE *
8500  DG = DG - 12: X = PX + SX + AX
8510  IF Q = 3 GOTO 8560
8520  IF Q = 4 GOTO 8570
8530  IF Q = 5 GOTO 8580
8540  Y = RND(400): IF Y > X PRINT"  ","GIBBLE!": GOTO 500
8550  GOTO 8150
8560  Y = RND(350): IF Y > X GOTO 7780 ELSE GOTO 8190
8570  Y = RND(500): IF Y > X GOTO 6670 ELSE GOTO 8190
```

```
8580  Y = RND(700): IF Y > X GOTO 9100 ELSE GOTO 8190
8599  REM * ROCK *
8600  DG = DG - 12: X = PX + SX + (AX/3)
8610  IF Q = 3 GOTO 8660
8620  IF Q = 4 GOTO 6670
8630  IF Q = 5 GOTO 8680
8640  PRINT"  ","GIBBLE!": GOTO 500
8660  Y = RND(300): IF Y > X GOTO 7780 ELSE GOTO 8190
8680  Y = RND(500): IF Y > X GOTO 9100 ELSE GOTO 8190
8699  REM * STICK *
8700  DG = DG - 15: X = SX + (AX/2) + (PX/2)
8710  IF Q = 3 GOTO 8760
8720  IF (Q = 4) OR (Q = 5) GOTO 8770
8730  Y = RND(350): IF Y > X PRINT"  ","GIBBLE!": GOTO 500
8740  GOTO 8150
8760  Y = RND(350): IF Y > X GOTO 7780 ELSE GOTO 8190
8770  PRINT"The stick snaps into pieces.": DG = DG - 2: J(8) = 0
8780  IF Q = 4 GOTO 6670 ELSE GOTO 9100
8799  REM * GUM *
8800  DG = DG - 6: X = AX + (SX/2) + (PX/10)
8810  IF Q = 3 GOTO 7780
8820  IF Q = 4 GOTO 6670
8830  IF Q = 5 GOTO 8860
8840  Y = RND(260): IF Y > X PRINT"  ","GIBBLE!": GOTO 500
8850  GOTO 8150
8860  Y = RND(200): IF Y > X GOTO 9100 ELSE GOTO 8190
8899  REM *SLIMY THING *
8900  IF Q = 3 GOTO 8950
8910  IF Q = 4 GOTO 6670
8920  IF Q = 5 GOTO 8190
8930  PRINT"  ","GIBBLE!": GOTO 500
8950  PRINT"The Brinchley Beast turns into a hungry kufu!"
8960  LC(L1,L2) = 4: EX(L1,L2) = 4: Q = 4: GOTO 6670
8999  REM * SWORD *
9000  X = PX + (SX/2) + (AX/2): DG = DG - 5
9010  IF Q = 3 GOTO 8190
9020  IF Q = 4 GOTO 9050
9030  IF Q = 5 GOTO 9070
9040  GOTO 8150
9050  Y = RND(333): IF Y > X GOTO 6670 ELSE GOTO 8190
9070  Y = RND(250): IF Y > X GOTO 9100 ELSE GOTO 8190
9100  PRINT"NONE TOO PLEASED WITH YOUR ATTITUDE,  ";
9110  PRINT"THE GRIMPH BREAKS YOUR  ";
9120  X = RND(10): IF X = 1 AND S(11) = 0 GOTO 9120
9130  IF X = 1 PRINT"COMPASS!": S(11) = 0
```

```
9140 IF X = 2 PRINT"ARM!": DG = DG – 15
9150 IF X = 3 PRINT"LEG!": DG = DG – 20
9160 IF X = 4 PRINT"NECK!": DG = DG – 70
9170 IF X = 5 PRINT"THUMBNAIL!": DG = DG – 2
9180 IF X = 6 PRINT"NOSE!": DG = DG – 10
9190 IF X = 7 PRINT"BIG TOE!": DG = DG – 3
9200 IF X = 8 PRINT"BACK!": DG = DG – 65
9210 IF X = 9 PRINT"FINGER!": DG = DG – 3
9220 IF X = 10 PRINT"SKULL!": DG = DG – 75
9230 GOTO 650
9300 IF K = 1 GOTO 9330
9310 K = 1: TB = 0: PRINT"  ","brinch-";: GOSUB 10000
9320 PRINT"LEY!!!": PRINT: GOSUB 10000
9330 PRINT"A Brinchley Beast is here.": GOTO 650
9350 IF K = 2 GOTO 9400
9360 PRINT"A hungry kufu blocks your path!": GOSUB 10000
9370 PRINT: PRINT"It salivates in foul anticipation of the meal to come!":
     PRINT
9380 K = 2: GOTO 9410
9400 PRINT"A kufu is here."
9410 X = RND(10): IF X > 6 GOTO 6670
9420 GOTO 650
9450 PRINT"A Grimph is here.": X = RND(10): IF X > 3 GOTO 9110
9460 GOTO 650
9500 IF K = 3 GOTO 9550
9510 GOSUB 10000: FOR X = 1 TO 3: Z = RND(25) + 1: FOR Y = 1 TO Z
9520 PRINT"  ";: NEXT Y: PRINT"gibble!  ";: NEXT X
9530 PRINT: PRINT: PRINT"A wild purofolee hops into view!": K = 3
9540 GOTO 650
9550 PRINT"A purofolee is here.": PRINT
9560 PRINT"PUROFOLEE: Gibble?": PRINT
9570 GOTO 650
9600 IF K = 4 GOTO 9650
9610 RV = RND(5): PRINT"You come to a river bank"
9620 PRINT: K = 4: GOTO 9660
9650 PRINT"You are at the bank of a river.": PRINT
9660 IF RV = 1"It is quite pleasant here."
9670 IF RV = 3 PRINT"The smell of ancient sewage is unpleasant, but bearable."
9680 DG = DG + .25: GOTO 650
9700 X = L1: Y = L2: IF Q$ = "N" THEN X = L1 – 1
9710 IF X < 1 THEN X = 10
9720 IF Q$ = "S" THEN X = L1 + 1
9730 IF X > 10 THEN X = 1
9740 IF Q$ = "E" THEN Y = L2 – 1
9750 IF Y < THEN Y = 10
```

```
 9760  IF Q$ = "W" THEN Y = L2 + 1
 9770  IF Y > 10 THEN Y = 1
 9780  IF X = L3 AND Y = L4 GOTO 9810
 9790  IF Q = 7 PRINT"THE RIVER   "; ELSE PRINT"THE MOUNTAIN   ";
 9800  PRINT"IS IN YOUR WAY.": DG = DG - 3: GOTO 650
 9810  L3 = L1: L4 = L2: L1 = X: L2 = Y: K = 0: GOTO 400
 9819  REM * WAIT *
 9820  PRINT"   ","(pause)": PRINT: GOSUB 10000
 9830  DG = DG + 10: IF DG > DX THEN DG = DX
 9840  GOTO 500
 9850  PRINT"Your arms are full. You can't carry anything more unless"
 9860  PRINT"you drop something.": DG = DG - 1: GOTO 500
 9899  REM * GET ALL *
 9900  IF L1 = R1 AND L2 = R2 GOTO 9920
 9910  GOTO 970
 9920  FOR X = 1 TO 12: IF A(X) = L1 AND B(X) = L2 GOTO 9950
 9930  NEXT: PRINT"SUPPLIES GATHERED.": GOTO 500
 9950  A(X) = 0: B(X) = 0: S(X) = 1: GOTO 9930
 9960  IF L1 = R1 AND L2 = R2 GOTO 9980
 9970  PRINT"You don't have jet propulsion engines in your shoes!":
       DG = DG - 2: GOTO 500
 9980  FOR X = 1 TO 100: PRINT"  *  ";: FOR Y = 1 TO 55: NEXT: NEXT
 9985  PRINT: PRINT: GOSUB 10400
 9990  PRINT"Your score was  ";SC: IF SC: IF SC > 75 PRINT"FANTASTIC
       WORK,  ";N$;"!"
 9995  IF SC < 25 PRINT"I'm not too impressed by your performance. . ."
 9997  END
 9999  STOP
10000  FOR TT = 1 TO 321: NEXT: RETURN
10010  PRINT: PRINT"YOUR MISSION,  ";N$;"  IS TO EXPLORE THE"
10020  PRINT"MYSTERIOUS PLANET MARS!": PRINT
10030  PRINT"An ancient civilization once thrived here. You must find as"
10040  PRINT"many of the treasured relics from this long-dead culture as"
10050  PRINT"you can, return them to your rocket ship, and blast off for"
10060  PRINT"good old Earth.": PRINT
10070  PRINT"But watch out—some items you will find may be worthless"
10080  PRINT"junk that will have a negative effect on your score."
10090  PRINT"Some of the junk might, however, come in handy during your"
10100  PRINT"exploration of the Red Planet": PRINT
10110  INPUT"Please press 'ENTER' for more  ";Q$
10120  CLS: PRINT: PRINT"You will encounter anumber of Martian beasts and"
10130  PRINT"monsters during your travels. Different types of creatures"
10140  PRINT"must be handled in different ways. The successful methods may"
10150  PRINT"not always be obvious. The bizarre natural forces of Mars"
10160  PRINT"will also tend to present obstacles. ": PRINT: PRINT
```

```
10170 PRINT"You may define the characteristics of your explorer by setting"
10180 PRINT"a value from 1 to 100% for each quality (such as strength,  "
10190 PRINT"speed, etc.), or you can let the computer automatically define"
10200 PRINT"your character for you.": PRINT: INPUT"Please press 'ENTER' for
      more  ";Q$
10210 CLS: PRINT: PRINT"During the game, each command may be one or two
      words, such as;"
10220 PRINT"LOOK", "DROP ROCK", "EAT FOOD", "WAIT"
10230 PRINT"Or a single letter directional move command;"
10240 PRINT"(N = North, S = South, E = East, and W = West)"
10250 PRINT"Commands consisting of more than two words, such as;"
10260 PRINT"PUT KUFU IN POCKET  ","are not valid."
10270 PRINT: PRINT"If you have no idea of what to do, you may use the special"
10280 PRINT"command HELP for a list of commands used in the game."
10290 PRINT"NOT ALL COMMANDS WILL BE ACCEPTABLE UNDER ALL
      CIRCUMSTANCES!"
10300 PRINT"Also, what may be helpful at one time, may be of no particular"
10310 PRINT"effect at another, and actually harmful at a third time."
10320 PRINT"For instance, EATing in front of a Grimph is not recommended."
10330 PRINT: INPUT"Please press 'ENTER' to define your character  ";Q$
10340 RETURN
10399 REM * SCORE CALCULATION *
10400 SC = 0: FOR X = 1 TO 12: IF T(X) = 1 THEN SC = SC + 10
10410 IF E(X) = R1 AND F(X) = R2 THEN SC = SC + 12
10420 IF J(X) = 1 THEN SC = SC - 2
10430 IF C(X) = R1 AND D(X) = R2 THEN SC = SC - 3.5
10440 NEXT: SC = SC + SV: RETURN
10449 REM * SUPPLIES PRESENT *
10450 IF X = 1 PRINT"Some food is here."
10460 IF X = 2 PRINT"A bottle of water is here."
10470 IF X = 3 PRINT"A knife is here."
10480 IF X = 4 PRINT"A gun is here."
10490 IF X = 5 PRINT"A laser is here."
10500 IF X = 6 PRINT"A coil of rope is here."
10510 IF X = 7 PRINT"An inflatable raft is here."
10520 IF X = 8 PRINT"A flashlight is here."
10530 IF X = 9 PRINT"A metal pipe is here."
10540 IF X = 10 PRINT"Some old magazines are here."
10550 IF X = 11 PRINT"A compass is here."
10560 IF X = 12 PRINT"Your spacesuit is hanging neatly on its rack."
10565 Y = Y + 1: RETURN
10569 REM * JUNK PRESENT *
10570 IF X = 1 PRINT"An old shoe is here."
10580 IF X = 2 PRINT"A gaudily ornate ring is here."
10590 IF X = 3 PRINT"A rock is here."
```

```
10600 IF X = 4 PRINT"A pair of fossilized undershorts is here."
10610 IF X = 5 PRINT"A large clot of dirt is here."
10620 IF X = 6 PRINT"An old bone is here."
10630 IF X = 7 PRINT"A sharpened stick is here."
10640 IF X = 8 PRINT"A badly chipped urn is here."
10650 IF X = 9 PRINT"A petrified wad of bubble gum is here."
10660 IF X = 10 PRINT"A colorful flower is here."
10670 IF X = 11 PRINT"A dead butterfly is here."
10680 IF X = 12 PRINT"An indescribable slimy thing is here."
10690 Y = Y + 1: RETURN
10699 REM * TREASURES PRESENT *
10700 IF X = 1 PRINT"A dented copper bowl is here."
10710 IF X = 2 PRINT"Some gold coins are here."
10720 IF X = 3 PRINT"A fossilized slide rule is here."
10730 IF X = 4 PRINT"A statue of a three-armed Martian god is here."
10740 IF X = 5 PRINT"A tarnished silver cup is here."
10750 IF X = 6 PRINT"A glass orb is here."
10760 IF X = 7 PRINT"A scroll inscribed with the ancient Martian language is
      here."
10770 IF X = 8 PRINT"Some glittering stones are here."
10780 IF X = 9 PRINT"A mysteriously humming box is here."
10790 IF X = 10 PRINT"A large, polished sword is here."
10800 IF X = 11 PRINT"A bleached skull is here."
10810 IF X = 12 PRINT"A set of blueprints for an ancient Martian palace is here."
10820 Y = Y + 1: RETURN
10849 REM * DEAD MONSTER *
10850 PRINT"There is a dead  ";
10860 IF Q = -1 PRINT"Squeanley serpent";
10870 IF Q = -2 PRINT"ghost of an Ancient Martian";
10880 IF Q = -3 PRINT"Brinchley Beast";
10890 IF Q = -4 PRINT"kufu";
10900 IF Q = -5 PRINT"Grimph";
10910 IF Q = -6 PRINT"purofolee";
10920 PRINT"  here.": X = RND(10): IF (X > 7) OR (Q = -3) OR (Q = -5)
      PRINT"The smell is horrendous!"
10930 RETURN
10949 REM * SERPENT *
10950 PRINT"DARN! You were just bitten by a Squeanly serpent!"
10960 DG = DG - 5: RETURN
10999 REM * GHOST *
11000 PRINT"A ghost of an Ancient Martian suddenly appears before you!"
11010 X = RND(10): Y = RND(10): Z = LC(X,Y)
11020 IF Z < 1 OR Z = 2 GOTO 11120
11030 IF Z > 6 GOTO 11010
11040 PRINT"'Lo,' sayith the ghost, 'at  ";X;":";Y;'  you shall find a  ";
```

```
11050 IF Z = 1 PRINT"Squeanly serpent"
11060 IF Z = 3 PRINT"Brinchley Beast"
11070 IF Z = 4 PRINT"kufu"
11080 IF Z = 5 PRINT"Grimph"
11090 IF Z = 6 PRINT"purofolee"
11100 EX(X,Y) = Z
11110 PRINT"The ghost vanishes into thin air!": LC(L1,L2) = 0: EX(L1,L2) = 0:
      RETURN
11120 PRINT"  ","BOO!": PRINT: GOTO 11110
11149 REM * DIA *
11150 X = DX*.8: IF DG > X PRINT"You're feeling just fine & dandy!":
      RETURN
11160 IF DG > 150 PRINT"You feel very good.": RETURN
11170 IF DG > 120 PRINT"You feel pretty good.": RETURN
11180 IF DG > 105 PRINT"You're not feeling too bad, all things considered.":
      RETURN
11190 IF DG > 90 PRINT"Some Alka-Seltzer might be nice. . .": RETURN
11200 IF DG > 80 PRINT"You've had better days.": RETURN
11210 IF DG > 70 PRINT"You're in no shape to go dancing.": RETURN
11220 IF DG > 60 PRINT"You're feeling rather poorly.": RETURN
11230 IF DG > 50 PRINT"Are your insurance payments up to date?": RETURN
11240 IF DG > 40 PRINT"Better rehearse your moans and groans.": RETURN
11250 IF DG > 30 PRINT"You're not doing well at all.": RETURN
11260 PRINT"It's a wonder you can still stand up!": RETURN
11299 REM * INVENTORY *
11300 PRINT: PRINT"YOU ARE NOW CARRYING—  "
11310 IF S(1) = 1 PRINT"FOOD  ",
11320 IF J(2) = 1 PRINT"ORNATE RING  ",
11330 IF T(3) = 1 PRINT"FOSSILIZED SLIDE RULE  ",
11340 IF J(4) = 1 PRINT"FOSSILIZED UNDERSHORTS  ",
11350 IF S(5) = 1 PRINT"LASER  ",
11360 IF J(6) = 1 PRINT"OLD BONE  ",
11370 IF T(7) = 1 PRINT"SCROLL  ",
11380 IF J(8) = 1 PRINT"URN  ",
11390 IF S(9) = 1 PRINT"METAL PIPE  ",
11400 IF J(10) = 1 PRINT"FLOWER  ",
11410 IF T(11) = 1 PRINT"SKULL  ",
11420 IF J(12) = 1 PRINT"SLIMY THING (?)  ",
11430 IF S(11) = 1 PRINT"COMPASS  ",
11440 IF T(10) = 1 PRINT"LARGE SWORD  ",
11450 IF J(9) = 1 PRINT"PETRIFIED WAD OF BUBBLE GUM  ",
11460 IF T(8) = 1 PRINT"GLITTERING STONES  ",
11470 IF S(7) = 1 PRINT"INFLATABLE RAFT  ",
11480 IF T(6) = 1 PRINT"GLASS ORB  ",
11490 IF J(5) = 1 PRINT"CLOT OF DIRT  ",
```

```
11500 IF T(4) = 1 PRINT"STATUE OF MARTIAN GOD  ",
11510 IF S(3) = 1 PRINT"KNIFE  ",
11520 IF T(2) = 1 PRINT"GOLD COINS  ",
11530 IF J(1) = 1 PRINT"OLD SHOE  ",
11540 IF T(1) = 1 PRINT"COPPER BOWL  ",
11550 IF S(2) = 1 PRINT"BOTTLE OF WATER  ",
11560 IF S(2) = 2 PRINT"EMPTY BOTTLE  ",
11570 IF J(3) = 1 PRINT"ROCK  ",
11580 IF S(4) = 1 PRINT"GUN  ",
11590 IF T(5) = 1 PRINT"SILVER CUP  ",
11600 IF S(6) = 1 PRINT"COIL OF ROPE  ",
16100 IF J(7) = 1 PRINT"SHARPENED STICK  ",
11620 IF S(8) = 1 PRINT"FLASHLIGHT  ",
11630 IF T(9) = 1 PRINT"MYSTERIOUSLY HUMMING BOX  ",
11640 IF S(10) = 1 PRINT"OLD MAGAZINES  ",
11650 IF J(11) = 1 PRINT"BUTTERFLY  ",
11660 IF T(12) = 1 PRINT"BLUEPRINTS  ",
11670 IF S(12) = 1 PRINT"SPACESUIT  ",
11680 Z = 0: FOR Y = 1 TO 12: Z = Z + S(Y) + J(Y) + T(Y): NEXT: IF Z = 0
      PRINT"nothing",
11690 PRINT: RETURN
11699 REM * ERROR *
11700 X = RND(8): IF X = 1 PRINT"SAY WHAT?"
11710 IF X = 2 PRINT"That does not compute."
11720 IF X = 3 PRINT"DON'T BE SILLY,  ";N$;"!"
11730 IF X = 4 PRINT"???!???"
11740 IF X = 5 PRINT"I don't understand."
11750 IF X = 6 PRINT"How would that help here?"
11760 IF X = 7 PRINT"What on Mars are you babbling about,  ";N$;"?"
11770 IF X = 8 PRINT"THAT DOESN'T MAKE ANY SENSE,  ";N$;"!"
11780 RETURN
11799 REM * HELP *
11800 PRINT: PRINT"POSSIBLE COMMANDS INCLUDE—  "
11810 PRINT"BLAST OFF", "CLIMB", "CRY", "DIA (DIAGNOSIS)",
      "DRINK",
11820 PRINT "DROP","EAT","GET","HELP","INFLATE",
11830 PRINT"INV (inventory)", "KILL", "LOOK", "MAP", "OPEN",
11840 PRINT"PRAY", "SCORE", "WAIT"
11850 PRINT: PRINT"Not all commands will be recognized at all times."
11860 PRINT: INPUT"Please press 'ENTER' to return to the game  ";Q$
11870 RETURN
11899 REM * RAVINE *
11900 L3 = L1: L4 = L2: PRINT"YOU JUST FELL INTO A DEEP RAVINE!":
      DG = DG - RND(DG)
11910 FOR X = 1 TO 4: Y = RND(11): IF T(Y) = 1 GOTO 11980
```

```
11920 IF J(Y) = 1 GOTO 11990
11930 IF S(Y) > 0 GOTO 12000
11940 NEXT: PRINT: GOSUB 10000
11950 PRINT"It takes quite a bit of effort, but you manage to crawl out to"
11960 PRINT "the south.": PRINT:L1 = L1 + 1:IF L1>10 THEN L1 = 1
11970 Q = LC(L1,L2): EX(L1,L2) = Q: RETURN
11980 T(Y) = 0: E(Y) = RND(10): F(Y) = RND(10): GOTO 11940
11990 J(Y) = 0: C(Y) = RND(10): D(Y) = RND(10): GOTO 11940
12000 S(Y) = 0: A(Y) = RND(10): B(Y) = RND(10): GOTO 11940
12049 REM * MARSQUAKE *
12050 PRINT"The ground begins to rumble beneath your feet!": PRINT
12060 L1 = RND(5) + L1 - 3: IF L1>10 THEN L1 = 1
12070 IF L1 < 1 THEN L1 = 10
12080 L3 = L - 1: IF L3 < 1 THEN L3 = 10
12090 L2 = L2 + RND(5) - 3: IF L2 < 1 THEN L2 = 10
12100 IF L2 > 10 THEN L2 = 1
12110 L4 = L2: FOR X = 1 TO 40
12120 Y = RND(10): Z = RND(10): ZZ = LC(Y,Z): IF ZZ = 20 GOTO 12140
12130 IF ZZ > 0 THEN ZZ = - ZZ
12140 LC(Y,Z) = ZZ: NEXT
12150 PRINT" ","* whew! *": PRINT"The Marsquake is over now!"
12160 LC(L1,L2) = 0: Q = 0: EX(L1,L2) = 0: RETURN
12199 REM * STORM *
12200 PRINT"You are caught in a weird Martian storm!": PRINT
12210 LC(L1,L2) = 0: Q = 0: EX(L1,L2) = 0
12220 DG = DG - RND(DG/4)
12230 FOR X = 1 TO 40: Y = RND(10): Z = RND(10)
12235 IF LC(Y,Z) = 20 GOTO 12250
12240 LC(Y,Z) = - LC(Y,Z)
12250 NEXT: GOSUB 10000
12260 PRINT"The weather seems to be clearing up now.": PRINT
12270 RETURN
12299 REM * SKY *
12300 PRINT"ODD . . . THE SKY TURNS A FUNNY COLOR FOR A FEW
      MINUTES . . ."
12310 PRINT: GOSUB 10000
12320 FOR X = 1 TO 25: Y = RND(10): Z = RND(10)
12330 ZZ = LC(Y,Z): IF ZZ = 20 GOTO 12360
12335 IF ZZ > 12 THEN ZZ = - 1
12340 LC(Y,Z) = ZZ + 1
12360 NEXT: PRINT"Well, everything seems to be back to normal now. ";
12370 GOSUB 10000: PRINT"I guess . . .": PRINT
12380 RETURN
12399 REM * MAP *
12400 CLS: PRINT: PRINT
```

```
12410  FOR X = 1 TO 10: PRINT"  ",: FOR Y = 1 TO 10
12415  IF L1 = X AND L2 = Y PRINT"+  ";: GOTO 12450
12420  Z = EX(X,Y): IF Z< 1 OR Z>9 GOTO 12440
12430  ON Z GOTO 12500, 12510, 12520, 12530, 12540, 12550, 12560, 12570,
       12580
12440  IF Z = 20 PRINT"*  "; ELSE PRINT".  ";
12450  NEXT: PRINT: NEXT: PRINT
12460  PRINT"+ = YOU, * = SHIP, S = SQUEANLY SEARPENT, B =
       BRINCHLEY BEAST,"
12470  PRINT"K = KUFU, G = GRIMPH, P = PUROFOLEE, R = RIVER, M =
       MOUNTAIN,"
12480  PRINT"r = RAVINE  PRESS 'ENTER' TO CONTINUE GAME  ";
12490  INPUT Q$: RETURN
12500  PRINT"S  ";: GOTO 12450
12510  PRINT".  ";: GOTO 12450
12520  PRINT"B  ";: GOTO 12450
12530  PRINT"K  ";: GOTO 12450
12540  PRINT"G  ";: GOTO 12450
12550  PRINT"P  ";: GOTO 12450
12560  PRINT"R  ";: GOTO 12450
12570  PRINT"M  ";: GOTO 12450
12580  PRINT"r  ";: GOTO 12450
12600  IF Q = 3 THEN MM = RND(4)
12610  IF Q = 4 THEN MM = RND(6)
12620  IF Q = 5 THEN MM = RND(10)
12630  IF Q = 6 THEN MM = RND(5)
12640 RETURN
```

Chapter 7

Graphics, Sound and Other Extras

You can create an infinite number of intriguing and creative games by using the described techniques. The programmer's imagination is the only limit to what can happen. However, this still may not be good enough.

All the game techniques presented in the previous chapters are text-oriented. The computer prints out a question; the player chooses and types in his answer; and the computer prints out the results. While this kind of game is fun, it does lack realism. Some players find it very difficult to really get into a purely text-oriented game.

This chapter takes a quick look at a few possible ways you can liven up an adventure game. These extras are not included in any of the complete game programs featured in this book because computer brands vary widely in capabilities and methods. Text-oriented BASIC commands are more universal, although there are some differences. Watch out for unfamilar commands if you try these programs on anything other than a TRS-80. Check the user's manual.

However, you should take full advantage of your particular computer's capabilities if you so choose.

Remember when adding these features to an adventure game that they are extras. A reflex-oriented game, like "Space Invaders" or "Pacman," is built primarily around graphic displays. In an adventure game, on the other hand, graphics enhance the realism and excitement of the game; but they should not be the main point of the game. This is important to remember while programming because both graphics (and many other special features) and basic adventure games each take up quite a bit of memory space. With a lengthy game program like "MARS" you may have no room left to add any graphics and should, unless your system has a very large memory.

It's usually a good idea to write the basic text-oriented game first, then go back and add the extras if you still have enough unused memory space.

Always run a complete sample before checking the memory size to allow for the storage of all the variable values. Strings and arrays can use a lot of memory space. If you fill too much memory space with actual programming, the game may bomb out with an "**OUT OF MEMORY**" error midway through.

Run several samples as you add extras. It's a good idea to leave at least 500 bytes open after a complete sample run just in case you didn't use all of the variables to their maximum limits in the test run. When the available memory size drops below this limit, stop programming. To add anything more will lead to trouble.

GRAPHICS

Graphics, probably the most popular computer extra, are simply computer-generated illustrations.

Programming methods for graphics vary widely from computer to computer.

One of the most common methods, crude but possible on virtually all microcomputers, is to print out selected alphanumeric characters in a specific pattern to form a rough image. Some commonly used characters for this trick are "X", ".", ":", "/", "o", "@", and "*". **Figure 7.1** shows a simple example. The drawings made by this technique are crude and awkward; but, in some cases, they may be sufficient. It is a novelty rather than a practical programming tool.

Most practical computer graphics systems are based on a grid matrix of dots (or individual screen positions) that can be independently turned on or off. Any specific point on the display screen may be specified by two coordinates (horizontal and vertical). For example, the TRS-80 uses a grid matrix of 128 horizontal dots and 48 vertical dots. The command **SET(8,5)** turns on the eighth horizontal dot in the fifth vertical column.

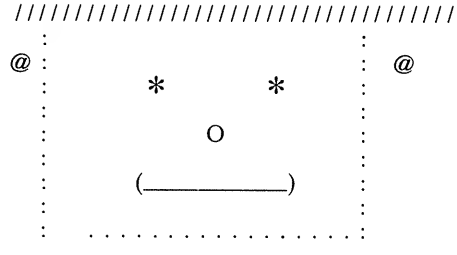
The greater the number of dots in such a grid matrix system, the greater the resolution of the graphics. That is, more dots allow greater detail and smoother appearing simulated arcs and diagonals. The 128 X 48 matrix of the TRS-80 is pretty coarse, and only marginally usable for representative illustrations.

Many modern microcomputers have far greater resolution. Many feature color capabilities. Each screen location may be turned on any of several colors. Color is often specified as a third coordinate. For example, **SET** (x,y,z) where x is the horizontal coordinate, y is the vertical coordinate, and z is the color identifier.

Some computers have more extensive graphics abilities and offer single commands to draw curves or lines at any specified location of the display screen. This can speed up the process noticeably.

Even so, the coordinate-plot system of graphics tends to be rather slow, which greatly limits the possibilities for anything beyond very crude animation (moving pictures).

Figure 7.1 Example of simple, typed graphics.



Many modern microcomputers feature special graphics characters that can be printed out like ordinary alphanumeric characters. This is somewhat faster than the coordinate-plot approach, but you must usually build the image line by line from top to bottom. This may or may not be a disadvantage, depending on the application. The **PRINT @** x command can alter the display sequence, but can be tricky unless the programmer is very careful and methodical.

Other computers have a specific range of memory addresses permanently dedicated to the monitor display screen. Each individual address contains the contents of a single specific screen location. You can **POKE** desired values into the appropriate memory addresses to create an illustration on the screen.

Unfortunately, BASIC is really too awkward and slow for really high quality graphics and animation. In most cases, you must resort to assembly (or machine) language for graphics routines, even if the main program is written in BASIC.

Most graphics generation systems are tedious and/or tricky to program, at least to some degree. It is usually necessary to go through several sample runs before the graphics illustrations come out exactly the way you want them. You can draw the desired image on graph paper first. But the picture often looks quite different once you get it on the monitor display screen.

APPLICATIONS FOR GRAPHICS

So how can you incorporate graphics into an adventure game? The programmer's imagination is the ultimate deciding factor. Possibilities are

virtually limitless. The following paragraph suggests a few typical applications.

You can display a drawing of the landscape whenever the player arrives at a new location (or commands "LOOK" or something similar). This landscape view may or may not be animated.

Rather than a verbal description, the items can be directly illustrated. Being told a kufu is looming over you is never quite the same as actually seeing it.

Some adventure games divide the monitor display screen into two (or more) discrete sections. One displays the standard text of the game, while the other section continuously illustrates the action of the game. In practical programs, due to memory limitations, these continuous illustrations generally tend to be rather crude. Often just simple geometric shapes are used. A triangle might represent the hero, a circle the treasure, and a square the monster. While far from being realistic, these graphics add a great feeling of action to the game. It is especially effective when used along with real-time techniques.

Perhaps you can invoke a special command ("SEE" for example) when the player meets a monster. This command would cause the computer to display a static, or simply animated picture of the creature, so the player can judge what he's up against. In the game of "MARS", for instance, the grimph should appear larger and more ferocious than a Brinchley Beast.

A programmer can incorporate some reflex-oriented tasks into an adventure game. In these situations, graphics are combined with what will be called real-time techniques. This is appropriate for battling monsters. The player has to dodge the monster's attack while he aims his own attack. This makes killing a monster more a test of skill than luck, as when the results are determined by randomly generated numbers.

Good graphic routines add considerable appeal to any game. But remember that the game itself should come first. Graphics consume mammoth portions of the computer's memory space. Don't scrimp on variety within the game to allow space for graphics for if the game wears thin, nobody will bother to play and the graphics won't be seen at all.

SOUND EFFECTS

Sound effects can also add a great deal to the excitement of a good adventure game. For example, a monster roars as it attacks, or screams as it dies. Perhaps the player can hear the hero's labored breath (in "MARS") as he crawls out of a ravine.

Musical effects are another possibility. A fanfare could be played when the player vanquishes a monster, or a dirge when the battle turns out the other way.

Many popular microcomputers have limited sound generation capabilities and can direct access from BASIC. Often this includes only musical-like tones, although some models have more varied sound effect possibilities.

With most microcomputers, the programmer can produce a limited range of sounds by switching the cassette output port on and off in a specific pattern. A cable is then connected from the output port to an amplifier and speaker. By programming the correct sequence of output signals, the programmer can generate a number of tones and sound effects.

Once again, BASIC is generally too slow. The programmer usually has to use assembly, or machine, language routines to create sound effects using this method.

Programming sound effects can often be extremely tedious, unless the computer can use specific BASIC commands to generate the desired effect. But it is often worth the trouble to enliven an adventure game.

Several IC's will accept a computer (digital) output signal and use it to control a variety of sound effects. Perhaps the most versatile and powerful of those now on the market are the AY-3-8910 and AY-3-8912 Programmable Sound Generators. These devices, manufactured by General Instruments, were designed specifically to create computer controlled sound effects and music.

Each chip generates up to three independent tones and a random noise signal (useful for explosions and percussive, or drum-like, sounds). By modifying and combining these four basic signals, a programmer can generate thousands of complex sounds.

A big advantage of using a device like the AY-3-8910/8912 is that it contains its own internal latches. This means the chip will remember the instructions from the computer until they are over-ridden by new instructions. The computer can perform other tasks while the Programmable Sound Generator is producing a sound. This can speed up complex programs quite a bit.

A 61 page data and applications manual for the AY-3-8910/8912 Programmable Sound Generator is available from General Instrument dealers for further reference. To repeat, this type of external sound generation device makes computer controlled sound effects much easier.

REAL-TIME INPUTS

Adventure games in this book use straight inputs. That is, the computer will ask a question, then wait patiently for the player to make up his mind and reply. It will wait three weeks, or three years, as long as your electric bills are paid on time. Until the "**RETURN**" or "**ENTER**" key is pressed, nothing will happen. The program simply stops until an input is received.

Many people find computer games much more exciting when time limits force them to think fast. They feel more involved in the situation described in the game. This is called real-time play. Action is continuous, rather than stop and start.

Most modern microcomputers have an alternative **INPUT** command that will not stop the program instantly. On the TRS-80 this command is **INKEY$**. Any single character (not whole words) may be entered. The "ENTER" or "RETURN" key does not have to be hit. The computer accepts the input as soon as a key is depressed.

As an example, here is a simple routine that mght be used for facing a monster:

```
800 PRINT "YOU ARE FACING A MUQUOLP!"
810 PRINT: PRINT "YOUR CHOICE OF ACTION?"
820 PRINT "A --- ATTACK"
830 PRINT "B --- FLEE"
840 PRINT "C --- CRY": PRINT
850 C$ = "":T = 1
860 C$ = INKEY$: T = T+1: IF T = 300 GOTO 950
870 IF C$ = "" GOTO 860
880 REM * COMMAND CHECK ROUTINE *
        . . .
950 REM * MONSTER ATTACK ROUTINE *
```

Real-time techniques are often used along with graphics and/or sound effects.

For this type of real-time command routine, either print a menu of possible commands (as in the example shown above) and/or make sure that no two commands begin with the same letter.

SAVE GAME

Adventure games tend to be fairly lengthy, and it is not always possible to play an entire game in one sitting. This makes a **SAVE GAME** feature highly desirable.

At any point throughout the game, the player may choose to **SAVE** the game. This would probably be programmed along with the rest of the commands. That is, each command check routine would have a new line added, so that it recognizes "**SAVE**" as a legitimate command. For instance, in the main command check routine in "MARS", we could add the following line:

```
775 IF CX$ = "SAV" GOTO 12000
```

When the player selects the command "**SAVE**", all current variable values are loaded onto a cassette or a floppy disc.

At the beginning of the game, the player should be asked if he wants to start a new game, or **LOAD** an old game. If **LOAD** is selected, the variable values are read back into the computer from the tape or disc. The game will immediately pick up where it left off.

For details on how to **SAVE** and **LOAD** variable values, see your computer's manuals. It is unfortunately beyond the scope of this book to discuss the various ways this is done with popular microcomputers.

If you're fairly new to programming, I'd advise you not to bother with **SAVE** and **LOAD** routines. They can be rather tricky if you're not 100 percent sure of what you're doing.

SUMMARY

This chapter gives little specific detail, mainly because the techniques for these extra features differ so much from brand to brand. The intent here was simply to suggest a few ideas that might spark up your own imagination when you're working on your own adventure games.

Remember though, these goodies are all frills. You must first create a solid game that will stand up on its own. If you can then make the game even better by adding graphics, or other extras—great! But fantastic graphics and sound effects are no substitute for a well-thought-out, exciting game.

Chapter 8

Treasure Hunt

One of the most popular and enduring types of adventure fantasies is a hunt for a buried pirate treasure. This idea lends itself very well to the adventure game format.

This game program ("TREASURE HUNT"), listed in Table 8.1, is the simplest one in this book. It fits easily into a 16K computer.

In this game you are the captain of a modern day pirate ship, seeking a treasure that was buried by the crew of the Jolly Roger between 1600 and 1899. (The date is randomly selected in line 280).

The treasure is buried on one of ten islands in the area of a coral reef. A mermaid lives on the coral reef, and if you are lucky she may tell you that the treasure is not located on a specific island. However, apporaching the coral reef is terribly risky, and it may cost you one of your crew members, or even sink your ship and end the game.

As the game begins, you are sailing with a six man crew. Your crew members are:

ABRAMS BENNETT CLANCY DAWSON
EGBERT FRED

Any of these characters may be sent ashore to explore an island. The captain (you) can never leave the ship.

While a man is ashore on one of the islands, a simple map is displayed. A typical island map is illustrated in Figure 8.1.

Figure 8.1 Typical Island Map.

```
                X . T . .
                . . . R Q
T = Tree        . T . . .
R = Rock        . . . . R
Q = Quicksand   R . . T .
X = Your Man
```

Table 8.1 Complete Treasure Hunt Program.

```
 10 CLEAR 200
 20 REM * TREASURE HUNT * Delton T. Horn *
 30 DIM A(10,35): DIM B(35)
 40 CLS: PRINT: PRINT: PRINT"  ","THE PIRATES' TREASURE HUNT"
 50 PRINT: PRINT: INPUT"YOUR NAME, MATEY";N$
 60 FOR J = 1 TO 10: FOR K = 1 TO 25: Y = RND(17): IF 7 > 7 THEN Y = 1
 70 PRINT"  *  ";: IF Y > 4 THEN Y = Y - 4
 80 A(J,K) = Y: NEXT: NEXT
 90 CLS: PRINT: X = RND(10): XZ = X
100 Y = RND(25): A(X,Y) = 100: FOR Z = 1 TO 3
110 Y = RND(25): IF A(X,Y) > 1 GOTO 110
120 A(X,Y) = 10: NEXT
130 Y = RND(25): IF A(X,Y) > 3 GOTO 130
135 A(X,Y) = 20
140 FOR X = 1 TO 10
150 Y = RND(35): IF A(X,Y) > 1 GOTO 150
160 A(X,Y) = 10
170 Y = RND(30): IF A(X,Y) > 1 GOTO 170
180 A(X,Y) = 30: NEXT
190 X = RND(10): IF X = XZ GOTO 190
200 Y = RND(25): IF A(X,Y) > 1 GOTO 200
210 A(X,Y) = 40
220 AD = 3: BN = 3: CL = 3: DW = 3: EG = 3: FR = 3: FL = 0
230 S = RND(5): IF S = 1 S$ = "SOLEMN ROGER"
240 IF S = 2 S$ = "LEAKY TUB"
250 IF S = 3 S$ = "SEA DEVIL"
260 IF S = 4 S$ = "SON OF JOLLY ROGER"
270 IF S = 5 S$ = "PIRATE SHIP"
280 YY = (RND(3) + 15)*100: YR = RND(100) - 1 + YY
290 PRINT: PRINT"YOU ARE CAPTAIN  ";N$;","
300 PRINT"BLOOD-THIRSTY MASTER OF THE SS  ";S$
310 PRINT: PRINT"You are seeking the treasure buried by the pirate crew of
    the"
320 PRINT"'Jolly Roger' in  ";YR
330 PRINT: PRINT"  ","Your crew consists of—  "
340 IF AD > 0 PRINT"Abrams",
350 IF BN > 0 PRINT"Bennett",
360 IF CL > 0 PRINT"Clancy",
370 IF DW > 0 PRINT"Dawson",
380 IF EG > 0 PRINT"Egbert",
390 IF FR > 0 PRINT"Fred",
395 PRINT: IF FL = 1 PRINT"You are carrying the flag of the original Jolly
    Roger"
```

```
400 C = AD+BN+CL+DW+EG+FR: IF(C = 0) OR (C<0) GOTO 1000
410 PRINT: PRINT"THERE ARE 10 ISLANDS IN THE AREA (enter '11' to
    visit the"
420 PRINT"coral reef)": PRINT"WHICH ISLAND SHALL THE  ";S$;
430 INPUT" VISIT NEXT";I
440 I = INT(I): IF (I>0) AND (I<11) GOTO 480
450 IF I = 11 GOTO 3000
460 PRINT"What in the name of Bluebeard are you talking about,Cap'n?!?"
470 GOTO 410
480 PRINT: PRINT"Crew member to go ashore on island #";I;" *****"
490 IF AD>0 PRINT"  ","1—ABRAMS"
500 IF BN>0 PRINT"  ","2—BENNETT"
510 IF CL>0 PRINT"  ","3—CLANCY"
520 IF DW>0 PRINT"  ","4—DAWSON"
530 IF EG>0 PRINT"  ","5—EGBERT"
540 IF FR>0 PRINT"  ","6—FRED"
550 PRINT: PRINT"YOUR ORDER, CAPTAIN  ";N$;
560 INPUT C: IF (C = 1) AND (AD>0) GOTO 630
570 IF (C = 2) AND (BN>0) GOTO 630
580 IF (C = 3) AND (CL>0) GOTO 630
590 IF (C = 4) AND (DW>0) GOTO 630
600 IF (C = 5) AND (EG>0) GOTO 630
610 IF (C = 6) AND (FR>0) GOTO 630
620 PRINT: PRINT"  ","WHO??";: GOTO 560
630 CP = 1: FOR X = 1 TO 25: B(X) = A(I,X): NEXT
640 IF CP = 0 PRINT"CAPTAIN  ";N$;"  OF THE SS  ";S$: GOTO 330
650 GOSUB 2010
660 PRINT"  is ashore on island #";I: PRINT
670 GOSUB 2080
680 GOSUB 2010
690 PRINT":  Which way should I go, Cap'n";
700 INPUT D$: D$ = LEFT$(D$,1)
710 IF C = 1 THEN AD = AD – .1
720 IF C = 2 THEN BN = BN – .1
730 IF C = 3 THEN CL = CL – .1
740 IF C = 4 THEN DW = DW – .1
750 IF C = 5 THEN EG = EG – .1
760 IF C = 6 THEN FR = FR – .1
770 GOSUB 2210
780 IF D$ = "R" THEN CP = 0
790 IF CP = 0 GOTO 640
800 IF D$ = "N" GOTO 2500
810 IF D$ = "S" GOTO 2510
```

```
820  IF D$ = "E" GOTO 2520
830  IF D$ = "W" GOTO 2530
840  GOSUB 2010
850  PRINT": WHAT?!?"
860  GOSUB 2000
870  PRINT: PRINT"North, South, East, West, or Return to ship";
880  GOTO 700
909  REM * SPACE CHECK *
910  IF B(CP) = 100 GOTO 1240
920  IF B(CP) = 2 GOTO 1300
930  IF B(CP) = 3 GOTO 1430
940  IF B(CP) = 4 GOTO 1460
950  IF B(CP) = 10 GOTO 1520
960  IF B(CP) = 20 GOTO 1560
970  IF B(CP) = 30 GOTO 1620
980  IF B(CP) = 40 GOTO 1750
990  GOTO 640
999  STOP
1000 PRINT"no one": GOSUB 2000
1010 PRINT: PRINT: PRINT"Without a crew to guide her, your ship drifts"
1020 PRINT"helplessly across the Seven Seas throughout the"
1030 PRINT"rest of Eternity.": GOSUB 2000
1040 PRINT: PRINT" ","YOU LOSE, ";N$; PRINT: GOSUB 2000
1050 PRINT"Incidentally, the treasure was buried on island #";XZ
1060 PRINT: PRINT: PRINT: INPUT"Would you like to play again";S$
1070 S$ = LEFT$(S$,1): IF S$ = "Y" GOTO 40
1080 PRINT"OK. Good-bye."
1090 END
1099 REM * DROWNING *
1100 PRINT: GOSUB 2010
1110 PRINT": blub, blub, blub. . .": PRINT: PRINT: PRINT
1120 IF C = 1 THEN AD = 0
1130 IF C = 2 THEN BN = 0
1140 IF C = 3 THEN CL = 0
1150 IF C = 4 THEN DW = 0
1160 IF C = 5 THEN EG = 0
1170 IF C = 6 THEN FR = 0
1180 GOSUB 2000: GOSUB 2010
1190 PRINT" walked off the edge of the island, as ordered."
1200 PRINT"Of course, he drowns."
1210 GOSUB 2000
1220 INPUT"Please press 'ENTER' ";A$
1230 CP = 0: GOTO 640
1240 PRINT: GOSUB 2010
1250 PRINT" just found the treasure!!": PRINT
```

```
1260  GOSUB 2000
1270  PRINT"You and your entire remaining crew are now independently"
1280  PRINT"wealthy!!": PRINT
1290  GOSUB 2000: GOTO 1060
1300  PRINT: GOSUB 2010
1310  PRINT"  just ran into a tree.": PRINT
1320  IF C = 1 THEN AD = AD - .125
1330  IF C = 2 THEN BN = BN - .125
1340  IF C = 3 THEN CL = CL - .125
1350  IF C = 4 THEN DW = DW - .125
1360  IF C = 5 THEN EG = EG - .125
1370  IF C = 6 THEN FR = FR - .125
1380  GOSUB 2010
1290  PRINT":  OUCH!": PRINT
1400  INPUT"Please press 'ENTER'  ";A$
1410  GOSUB 2210
1420  GOTO 640
1430  PRINT: GOSUB 2010
1440  PRINT"  just tripped over a large rock!": PRINT
1450  GOTO 1320
1460  PRINT: GOSUB 2010
1470  PRINT"  just stepped into a pool of quicksand!": PRINT
1480  GOSUB 2000: PRINT
1490  GOSUB 2010: PRINT":  ARGH!!!!": PRINT
1500  GOSUB 2280
1510  GOTO 1400
1520  PRINT: GOSUB 2010
1530  PRINT"  just found a bleached human skull.": PRINT: GOSUB 2000
1540  GOSUB 2010: PRINT":  ick!": PRINT
1550  GOTO 1400
1560  IF FL = 1 GOTO 1800
1570  PRINT: PRINT"A ghost kills  ";
1580  GOSUB 2010: PRINT"!": PRINT
1590  X = RND(25): IF B(X) > 10 GOTO 1590
1600  A(I,X) = 20
1610  GOTO 1400
1620  PRINT: GOSUB 2010
1630  PRINT"  just ran into a hungry cannibal!": PRINT
1640  DC = RND(0)*2: IF C = 1 THEN AD = AD - DC
1650  IF C = 2 THEN BN = BN - DC
1660  IF C = 3 THEN CL = CL - DC
1670  IF C = 4 THEN DW = DW - DC
1680  IF C = 5 THEN EG = EG - DC
1690  IF C = 6 THEN FR = FR - DC
1700  GOSUB 2210
```

```
1710  GOSUB 2000: IF CP = 0 GOTO 1740
1720  PRINT"He just barely manages to escape with his life!"
1730  GOTO 1400
1740  PRINT: PRINT"Cannibal:  Burp!": PRINT: GOTO 1400
1750  PRINT: GOSUB 2010
1760  PRINT"  just found the flag of the old Jolly Roger!": PRINT
1770  FL = 1: GOSUB 2000
1780  B(CP) = 1: A(I,CP) = 1
1790  GOTO 1400
1800  PRINT:PRINT"The ghosts on the island all flee from the flag of"
1810  PRINT"the Jolly Roger!": PRINT
1820  FOR X = 1 TO 25
1830  IF B(X) = 30 THEN B(X) = 1
1840  IF A(I,X) = 30 THEN A(I,X) = 1
1850  NEXT: GOTO 1400
2000  FOR TT = 1 TO 345: NEXT: RETURN
2010  IF C = 1 PRINT"Abrams";
2020  IF C = 2 PRINT"Bennett";
2030  IF C = 3 PRINT"Clancy";
2040  IF C = 4 PRINT"Dawson";
2050  IF C = 5 PRINT"Egbert";
2060  IF C = 6 PRINT"Fred";
2070  RETURN
2079  REM * MAP *
2080  LC = 1: FOR X = 1 TO 5: PRINT"  ",: FOR Y = 1 TO 5
2090  IF CP = LC GOTO 2170
2100  IF B(LC) = 2 GOTO 2180
2110  IF B(LC) = 3 GOTO 2190
2120  IF B(LC) = 4 GOTO 2200
2130  PRINT".  ";
2140  LC = LC + 1: NEXT Y: PRINT: NEXT X
2150  PRINT: PRINT"T = Tree, R = Rock, Q = Quicksand, X = Your man": PRINT
2160  RETURN
2170  PRINT"X  ";: GOTO 2140
2180  PRINT"T  ";: GOTO 2140
2190  PRINT"R  ";: GOTO 2140
2200  PRINT"Q  ";: GOTO 2140
2210  IF AD < 0 PRINT"Abrams is dead": CP = 0: AD = 0: RETURN
2220  IF BN < 0 PRINT"Bennett is dead": CP = 0: BN = 0: RETURN
2230  IF CL < 0 PRINT"Clancy is dead": CP = 0: CL = 0: RETURN
2240  IF DW < 0 PRINT"Dawson is dead": CP = 0: DW = 0: RETURN
2250  IF EG < 0 PRINT"Egbert is dead": CP = 0: EG = 0: RETURN
2260  IF FR < 0 PRINT"Fred is dead": CP = 0: FR = 0: RETURN
2270  RETURN
2279  REM * KILL CHARACTER *
```

```
2280  IF C = 1 THEN AD = - 1
2290  IF C = 2 THEN BN = - 1
2300  IF C = 3 THEN CL = - 1
2310  IF C = 4 THEN DW = - 1
2320  IF C = 5 THEN EG = - 1
2330  IF C = 6 THEN FR = - 1
2340  RETURN
2499  REM * MOVE *
2500  CP = CP - 5: GOTO 2540
2510  CP = CP + 5: GOTO 2540
2520  CP = CP + 1: GOTO 2540
2530  CP = CP - 1
2540  IF (CP < 1) OR (CP > 25) GOTO 1100
2550  GOTO 910
3000  GOSUB 2000: SS = RND(15): IF SS = 13 GOTO 3100
3010  G = RND(8): IF (G = 1) AND (AD < 0.01) GOTO 3200
3020  IF (G = 2) AND (BN < 0.01) GOTO 3200
3030  IF (G = 3) AND (CL < 0.01) GOTO 3200
3040  IF (G = 4) AND (DW < 0.01) GOTO 3200
3050  IF (G = 5) AND (EG < 0.01) GOTO 3200
3060  IF (G = 6) AND (FR < 0.01) GOTO 3200
3070  IF G > 6 GOTO 3200
3080  PRINT: PRINT"In approaching the dangerous reef,  ";: C = G: GOSUB
      2010: PRINT"  is killed!"
3090  GOSUB 2280: GOTO 3200
3100  PRINT: PRINT"THE SS  ";S$;"  SINKS!": PRINT
3110  GOTO 1040
3200  PRINT: PRINT"A beautiful mermaid, who lives on the coral reef, tells you"
3210  H = RND(10): IF H = XZ GOTO 3210
3220  PRINT"the treasure is NOT buried on island #";H: PRINT
3230  INPUT"Please press 'ENTER'  ";A$
3240  GOSUB 2210
3250  CP = 0: GOTO 640
```

A character may pass through spaces marked as Rocks or Trees, but he will be injured. Stepping into quicksand is immediately fatal. But your crew is well-trained (albeit somewhat stupid). They will obey your orders and walk wherever you tell them to.

Each character has a health rating. Injuries (such as tripping over Rocks, or walking into Trees) decrease this health rating. If the health rating of a character drops to zero (or below), that character will die. If your entire crew is killed off, you lose the game.

Characters can also be killed off by having them walk off the north or south end of an island. The east/west ends loop around continuously to simplify the programming.

A character might also encounter a murderous ghost or a cannibal not displayed on the map. Once you encounter a ghost or cannibal it will not move. You must keep track of where ghosts or cannibals attack your men. The ghost will kill your man outright. He may, or may not, be able to escape the cannibal, depending on his health rating.

Characters encounter bleached human skulls, harmless (albeit disgusting) objects. The only island where more than one skull might appear is the one that contains the treasure. This can be a valuable clue.

The flag of the "Jolly Roger" is also hidden on one of the islands, but not the one with the treasure. Once you have this flag in your possession, all ghosts will flee.

The game ends when either you locate the treasure (you win), or kill off your entire crew (you lose).

PROGRAMMING

The TREASURE HUNT program is broken down into routines in Table 8.2. Table 8.3 lists all of the variables used in this program. The game is summarized in Table 8.4.

The game features 10 small islands. A map of each island is stored in an individual array. The 10 arrays are actually a single two-dimensional array. The first coordinate identifies the island. The second coordinate is the location on that island. For example, array location A(5,7) is position 7 on island 5.

A second, single dimension array is used to represent the island currently being visited. This array (B(35)) is used to display the island maps.

Each island is arranged in a 5 X 5 format, as shown in Figure 8.1. There are 25 spaces on each island. The arrays are dimensioned to 35 so that not all of the obstacles will appear on every island. Obstacles planted at array locations 26 through 35 are functionally invisible. This is somewhat wasteful of memory space, but that should not be a problem for this game, even with a computer with only 16K of memory.

After the string space is cleared (line 10) and the arrays dimensioned (line 30), the player's name is requested, and stored as N$.

Next, the island maps are preset. The values used to represent each object on the islands are summarized in Table 8.5.

Table 8.2 Routines and Subroutines Used in the "Treasure Hunt" Program.

Routines

10-50	initialize
60-220	preset variables
230-270	name ship

280-320	year treasure was buried
330-400	crew listing
410-470	island selection
480-620	crew member to go ashore selection
630	set current island
640-680	island display
690-880	input command
910-990	check contents of current location
1000-1050	entire crew dead
1060-1090	new game?
1100-1230	crew member drowns
1240-1290	treasure found
1300-1420	crew member runs into a tree
1440-1450	crew member trips over rock
1460-1510	crew member steps into quicksand
1520-1550	crew member finds skull
1560-1610	ghost attacks crew member
1620-1740	cannibal attacks crew member
1750-1790	crew member finds flag
1800-1850	ghosts flee from flag

Subroutines

2000	time delay
2010-2070	print crew member name
2080-2200	display island map
2210-2270	crew member dead announcement
2280-2340	crew member dies
2500-2550	calculate move
3000-3250	visit coral reef

In lines 60 through 80 each active island space (up to array location 25) is assigned a value from 1 to 4. A 1 is a blank location, a 2 is a tree, a 3 is a rock, and a 4 represents a pool of quicksand. The value for each space is assigned randomly. A random number from 1 to 17 is selected, and then manipulated to a value of 1, 2, 3, or 4. Table 8.6 shows the stored value for each possible value of Y. Notice that a 1 is the most likely value, and a 4 is the least likely.

A random value from 1 to 10 is assigned to X in line 90. This same value is also stored as XZ. This variable indicates which island holds the treasure. The treasure (value of 100) is planted randomly in one of the 25 active locations on the selected island in line 100.

Three skulls are randomly located on the treasure island (value of 10) in lines 100 through 120. When a location is randomly selected, if that location is not blank (the value is greater than 1), a new location is selected.

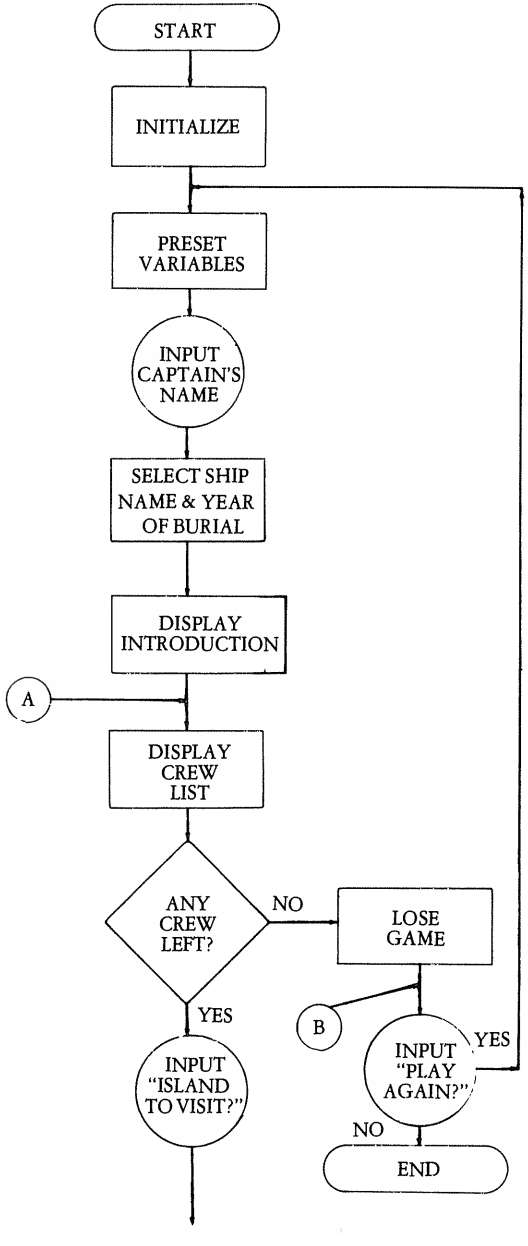


Figure 8.2A Flow Chart for TREASURE HUNT Program.

Table 8.3 Variables Used in the "Treasure Hunt" Program.

AD	Abrams' health rating
BN	Bennett's health rating
C	crew member to go ashore
CL	Clancy's health rating
CP	crew member's position on island
DC	effect of cannibal attack
DW	Dawson's health rating
EG	Egbert's health rating
FL	flag found?
FR	Fred's health rating
G	crew member killed by coral reef
H	mermaid's island choice
I	island to visit
J,K	preset counting loops
S	ship name select
SS	does ship sink at coral reef?
X,Y,Z	misc.
YR,YY	year of burial
	string variables
D$	directional command
N$	Captain's (player's) name
S$	ship name/play again?
	arrays
A(10,35)	main island maps
B(35)	current island map

A ghost (value of 20) is also placed on the island with the treasure in lines 130 and 135. If the random location contains the treasure, a skull, or a pool of quicksand (value greater than 3), a new location is selected.

In lines 140 through 180 a skull (10) and a cannibal (30) are placed on each of the 10 islands. Notice that this brings the total number of skulls on the island with the treasure (XZ) up to 4. These skulls may be placed at any array location up to 35. Skulls placed at locations 26 through 35 will not show up during the game. Similarly, cannibals may be placed anywhere from location 1 to 30, but only those in locations 1 to 25 will appear during the game. This way a skull or cannibal may or may not be found on each of the islands.

Finally, an island is selected for the flag (line 190). If the treasure island (XZ) is selected, a new island will be chosen. The flag may only be placed in one of the 25 active island locations. Its value is 40 (lines 200 and 210).

The preset routine is completed in line 220 where each of the six crew character health ratings (AD, BN, CL, DW, EG, and FR) are assigned an initial value of 3. The variable FL is preset to 0 to indicate that the flag has not been found.

Lines 230 through 270 randomly select one of five names for the player's pirate ship. You may substitute other names of your own creation. Alternatively, you could assign a permanent name that will be the same for every game, or substitute an **INPUT** statement allowing the player to select any name he likes for the ship.

A year is selected in line 280. A century is selected as YY. This variable may take on a value of 1600, 1700, or 1800. YR adds from 0 to 99 to this, making the year anything from 1600 to 1899. In the opening message (lines 290 through 320), this value is displayed as the year in which the treasure was buried. All of this is just for "show" and has no effect on the game itself. Touches like this can make the game more interesting.

Because this game is quite simple and self-prompting, no separate instruction routine is included in the program. If you want to include an instruction subroutine, you should call it at this point.

Lines 330 through 400 begin the actual game. The crew is displayed. Notice that the health rating of each crew member is checked, and the name displayed only if the crew member is alive (health rating greater than 0). For example:

```
340 IF AD>0 PRINT "Abrams",
```

Line 395 determines if the flag of the Jolly Roger has been found, and prints an appropriate message if it has.

In line 400 all of the crew health ratings are combined. If their combined value is 0 or less, the entire crew has been killed off, and the program jumps to the lose game routine at line 1000.

On the first round of the game, all six crew member names should be displayed.

Table 8.4 Summary of the "Treasure Hunt" Game.

Game location—Aboard a pirate ship in an area with ten islands and a coral reef

Hero/player character—Captain of the ship

Mission/Goal—To find the treasure hidden on one of the islands

Additional characters—The Captain's crew (Abrams, Bennett, Clancy, Dawson, Egbert, and Fred), Mermaid on coral reef (source of information), ghosts (obstacles), cannibals (obstacles)

Other obstacles—quicksand pits, approaching the coral reef, drowning, rocks (trip over), trees (walk into)

Table 8.5 Object Values for the "Treasure Hunt" Game.

1	blank space
2	tree
3	large rock
4	pool of quicksand
10	bleached human skull
20	ghost
30	cannibal
40	flag of the "Jolly Roger"
100	the treasure

THE PLAY

The player is prompted to select an island (1 through 10) to visit, or the coral reef (11) in lines 410 through 430. The player's input is checked for validity in lines 440 and 450. If an incorrect value has been entered, an error message (line 460) is displayed, and the player is asked for a new input. The player is not penalized for an incorrect entry.

Assume the player has entered "11" to visit the mermaid on the coral reef. The program jumps to line 3000. There is one chance in 15 (6⅔ percent) that the ship will sink (lines 3100 and 3110). Of course, this means the player loses the game.

If the ship is not sunk, a random value from 1 to 8 (G) is selected. If this value is 7 or 8, nothing special happens, and the program jumps ahead to line 3200. Each of the other six values represent each of the crewmen. If the selected crewman is not already dead, he will be killed.

In either case, the player selects a random number from 1 to 10 (H). If H is equal to XZ (the island with the treasure) a new value is selcted for H. Then a beautiful mermaid, who lives on the coral reef, tells you the treasure is NOT buried on island (H).

This information may, or may not be useful. It could allow you to eliminate that island from your search. On the other hand, the mermaid may tell you about an island you have already visited.

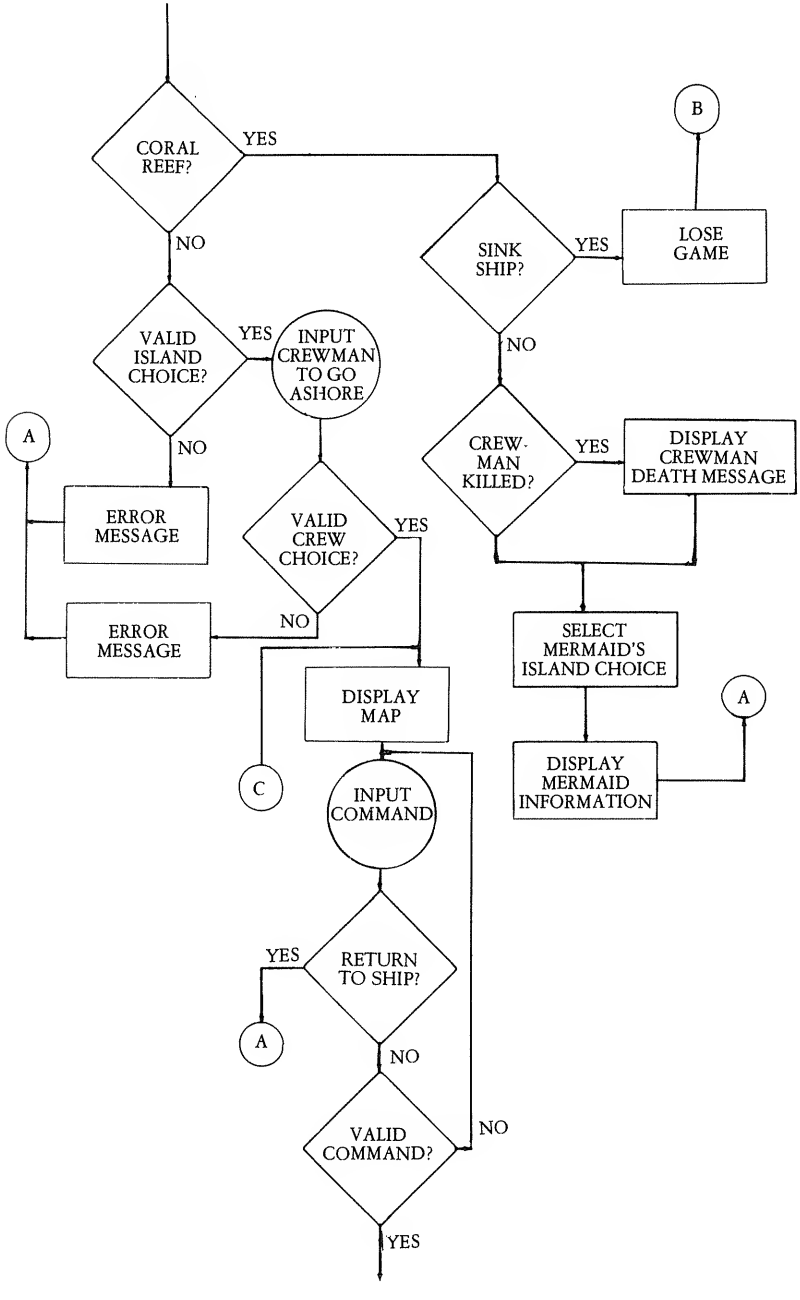


Figure 8.2B Flow Chart for TREASURE HUNT Program.

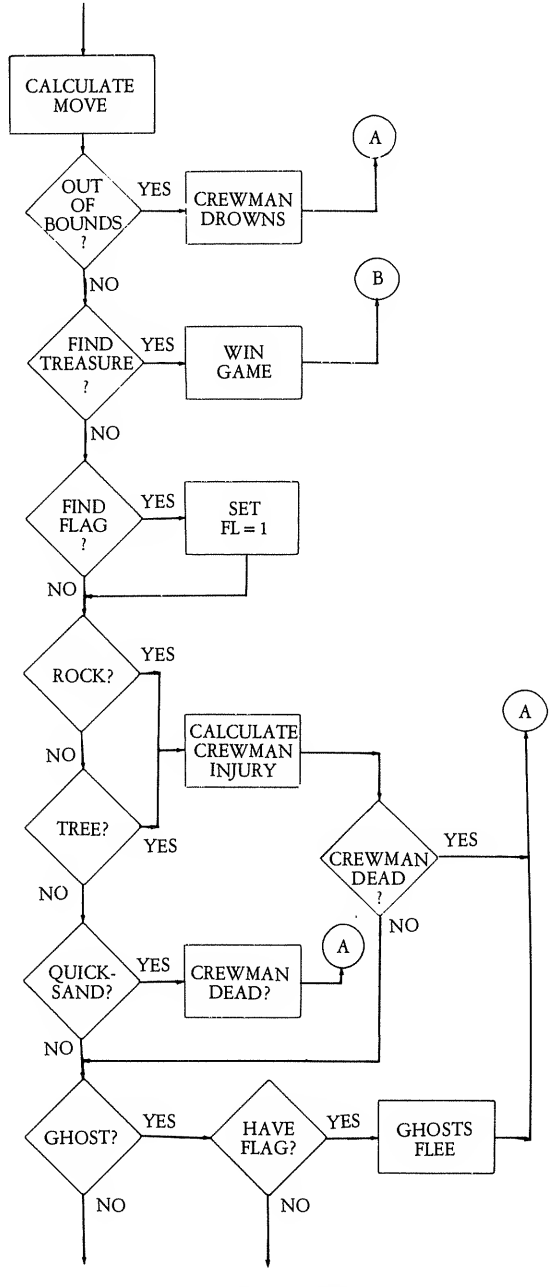


Figure 8.2C Flow Chart for TREASURE HUNT Program.

Visiting the coral reef is dangerous. There is a good chance that it will kill one of your crew. This means you should consult the mermaid sparingly. In any adventure game, frequent use of such advisors should be discouraged to prevent the game from getting too easy. If, for instance, there was no risk for visiting the coral reef, the player could keep going back until the mermaid identifies nine islands without the treasure. Then he visits the one remaining island to win the game unfairly.

On most moves, the player will select one island to visit. He is asked which crew member will be sent ashore. Living crew members only are acceptable (see lines 480 through 620). The number representing the selected crewman is stored as C.

The selected island map is put into temporary array B(x) (line 630), and the variable CP is set to equal 1. This variable is used to indicate the crewman's position on the island. Any visit to an island will always begin at array location 1. Any obstacle at this point (tree, rock, quicksand, ghost, or cannibal) will be ignored. If, however, the man is moved and then is sent back to position 1, the obstacle will behave in its normal manner.

A subroutine beginning at line 2080 displays the island map (as illustrated in Figure 8.1). The crewman then asks the Captain (the player) for his orders (lines 680 through 700). Only the first letter of the player's input (D$) is relevant. There are five acceptable commands:

N – move one space north (up)
S – move one space south (down)
E – move one space east (right)
W – move one space west (left)
R – return to the ship

The character may be returned to the ship from any position on the island.

An incorrect command will produce an error message, and a list of recognized commands (lines 840 through 880).

Notice that each command costs the crewman 0.1 from his health rating. This is true of both valid and invalid commands. Even if there were no obstacles, each character can only survive a total of 30 commands.

The contents of each location are checked in lines 910 through 990.

If the location has a value of 100, the treasure has been located, and the program jumps to line 1240 for the win game routine.

ENCOUNTERING OBSTACLES

If the map location has a value of 2, the player is informed that his crewman just ran into a tree (lines 1300 and 1310). That character's health rating is reduced by 0.125. Basically, the same thing happens when the

location has a value of 3, except the displayed message states that the crewman tripped over a large rock (lines 1430 through 1450).

A pool of quicksand is represented by a location value of 4. If you order your character to step into quicksand, he will be killed (lines 1460 through 1510).

Trees, rocks, and quicksand pools are displayed on each island map, so you can avoid them whenever possible. Occasionally, it may be necessary to pass a tree or a rock to get to another part of the island; but remember that your crew member will be injured. It is never good strategy to send a man into quicksand.

Other obstacle objects are not displayed on the island maps; but once found, they will not move (except the flag, and ghosts fleeing from the flag).

A location value of 10 represents a bleached human skull. A message is displayed that expresses the crew member's disgust (lines 1520 through 1550). Ordinarily, finding a skull has no direct effect on the game. However, if you find more than one skull on a single island, you can assume the treasure is hidden on that island.

If a ghost is encountered (location value is 20), the value FL (flag position) will be checked (line 1560). If FL equals 1, indicating the flag has been found, the program is diverted to line 1800, where all of the ghosts are removed from the appropriate island arrays. A message stating that all of the ghosts on the island flee from the flag of the Jolly Roger will be displayed.

Ghosts will only be encountered on the island with the treasure.

If FL equals 0, the flag has not been found. The ghost will immediately kill the crewman (lines 1570 and 1580), and his ghost is placed somewhere on the island (lines 1590 and 1600) for future haunting.

The game starts out with only one ghost, but you might end up with up to six haunting the treasure island (the original ghost and five dead crewmen). If the character is killed by anything other than a ghost, he will not return to haunt you.

The final obstacle is the cannibal (location value of 30). A random value of up to 2 (not necessarily an integer) is subtracted from the character's health rating (lines 1640 through 1700). If this value is enough to kill the character, the cannibal has a nice meal (see line 1740). Otherwise, the crewman manages to escape (line 1720), but he will be weakened by the encounter.

The location value might also be 40. This represents the flag of the Jolly Roger. The location value is set back to 1, and flag possession variable FL is set to 1 (lines 1750 through 1790).

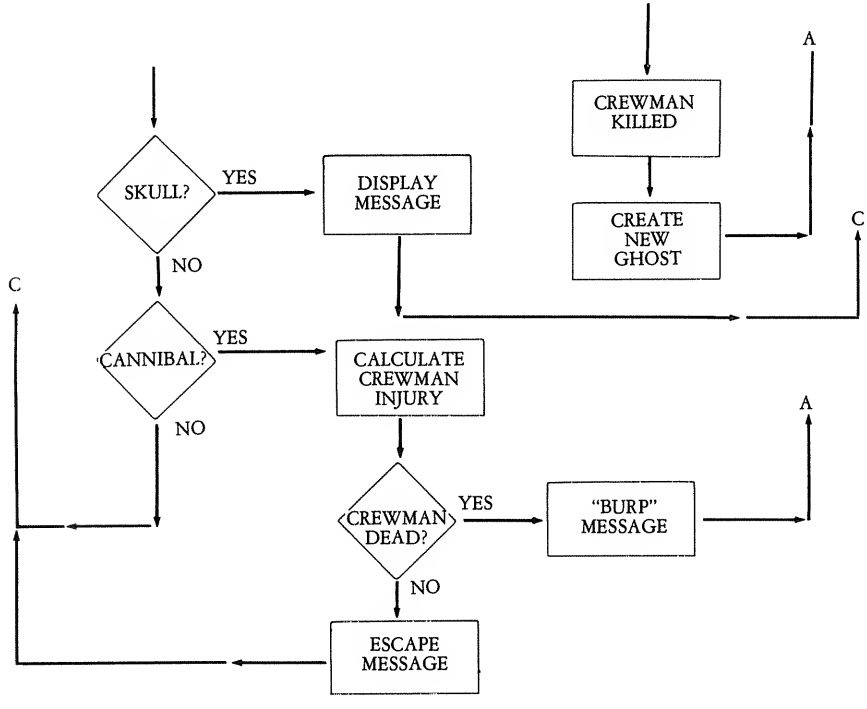


Figure 8.2D Flow Chart for TREASURE HUNT Program.

Table 8.6 Odds for Blank Spaces, Trees, Rocks, and Quicksand Pools in the "Treasure Hunt" Game.

(lines 60-80)

Y	adjusted value	object
1	1	blank
2	2	TREE
3	3	ROCK
4	4	QUICKSAND
5	1	blank
6	2	TREE
7	3	ROCK
8	1	blank
9	1	blank

10	1	blank
11	1	blank
12	1	blank
13	1	blank
14	1	blank
15	1	blank
16	1	blank
17	1	blank

		ODDS
1	blank	70.5%
2	TREE	12%
3	ROCK	12%
4	QUICKSAND	5.5%

POSSIBLE VARIATIONS

For an easier game, you could add more crew members, and/or increase their initial health ratings (line 220). For example, you might set the initial values at 4 or 5.

You can create a harder game by lowering the initial health ratings to 2, or 2.5. I do not recommend starting with health ratings lower than this because the game might become impossible to win.

For more variety, you could assign different health ratings for each of the characters. In this case, some of the men would die more readily than others. You could set the initial health ratings randomly, like this:

```
220 AD = RND(8)/2: BN = RND(8)/2: CL = RND(8)/2:
    DW = RND(8)/2: EG = RND(8)/2: FR = RND(8)/2:
    FL = 0
```

This line would give each character an initial health rating from 0.5 to 4.0 (in steps of 0.5). Some players may find this variation too frustrating.

SUMMARY

The TREASURE HUNT program is one of the simplest forms for an adventure game, but it is still plenty of fun to play. Included here, it demonstrates that you don't have to be an expert in fancy programming tricks, or spend months writing hundreds of lines, to create an enjoyable and worthwhile game program.

Chapter 9

The Golden Flute

Fantasy and fairy tales provide another profitable source for adventure game ideas. Elves, ogres, giants and gremlins all make fine characters in an adventure game. Adult fantasy has enjoyed a new popularity since *The Lord of the Rings* was published and such concepts are perfect for the adventure gamester.

This chapter will program a fantasy-oriented adventure game called "The Golden Flute". The story goes as follows.

Love is brought into the world when Brombiran, the Lord of the Woodland Elves, plays his magical Golden Flute. Unfortunately, the evil gremlin king, Terak, has stolen the Golden Flute. Now you must join the creatures of the Woodlands in a noble quest to recover the Golden Flute of Love.

PURPOSE

In this game the player must guide a Magic Chariot through the playing area in search of Terak's secret hideaway. Once the Golden Flute is in his possession, the player must return it safely to the Woodlands. The object of the game is to reach this goal in as few moves as possible. Naturally, along the way the player will encounter numerous monsters and obstacles. He may also find magical weapons and gold pieces.

The complete program listing for THE GOLDEN FLUTE is given in Table 9.1. Table 9.2 breaks the program down into its component routines.

CHARACTERS

The mythical creatures who accompany the player on the Quest to recover the Golden Flute could remain anonymous and identified only by numbers, or by type (e.g., ELF, SATYR, FAIRY, etc.). But the details of the fantasy are always part of the fun of a good adventure game.

In the program, I have provided the player with eight companions, including three elves, a satyr, two fairies, the Queen of the Sprites, and a princess. These characters are identified by name in Table 9.3.

Feel free to change any or all of the character names. They are listed at five points throughout the program (lines 732 through 822, 5442 through 5510, 5882 through 5450, 11010 through 11080, and 11390 through 11460). For consistency, all five of these lists should be updated for each name change.

Perhaps you prefer to let the player assign the character names at the beginning of the program, using string variables.

Table 9.1 Complete "Golden Flute" Program.

```
 10  REM * THE GOLDEN FLUTE * Delton T. Horn *
 19  REM * SET UP *
 20  CLEAR 100
 30  DIM A(100): DIM B(100): DIM C(9)
 40  CLS: PRINT: PRINT: PRINT"  ","THE GOLDEN FLUTE"
 50  PRINT: PRINT"  ","  ","by Delton T. Horn": PRINT: PRINT
 55  PW = 0: GW = 0
 60  INPUT"YOUR NAME";N$
 70  BR = 1: DG = 1: GR = 1: RU = 1: SJ = 1: JS = 1: AL = 1: PM = 1
 80  AX = 0: LC = 1: M = 1: SW = 0: FL = 0: MO = 0: GL = 0: MZ = 0
 90  FOR X = 1 TO 100: A(X) = 0: B(X) = 0: NEXT
 95  FOR X = 1 TO 9: C(X) = 1: NEXT
100  PRINT: PRINT"setting variables": PRINT: PRINT
109  REM * WOODS *
110  A(1) = 1: A(2) = 1: A(11) = 1: A(12) = 1
119  REM * ORACLE *
120  A(33) = 2
129  REM * PITS *
130  X = RND(7)+2: Y = RND(4)+4: X = X+Y*10: Y = X+1
140  A(X) = 3: A(Y) = 3: X = X+10: Y = Y+1: A(X) = 3: A(Y) = 3
149  REM * WALL *
150  Q = 0
160  Q = Q+1: X = (RND(4)+3)*10+RND(3)+3: Y = X+5: Z = 0
170  IF Q>7 GOTO 220
180  FOR V = X TO Y: IF A(V)>0 THEN Z = 1
190  IF A(V+10)>0 THEN Z = 1
200  NEXT: IF Z = 1 GOTO 160
210  FOR V = X TO Y: A(V) = 13: A(V+10) = 14: NEXT
220  FOR X = 1 TO 100: B(X) = A(X): PRINT"%";: NEXT: PRINT
229  REM * HIDDEN ITEMS *
230  X = RND(100)
```

```
232 IF B(X)>0 GOTO 230
235 B(X)=29
239 REM * TERAK & GUARDS *
240 X=(RND(4)+4)*10+RND(7)+2: IF B(X)>0 GOTO 240
250 B(X)=4: Z=9: Y=X-1: GOSUB 5010
260 Y=X+10: GOSUB 5010
270 Y=X-10: GOSUB 5010
280 Y=X+1: GOSUB 5010
290 Y=Y+10: GOSUB 5010
300 Y=Y-20: GOSUB 5010
309 REM * GARGOYLES *
310 U=RND(10)+2: FOR X=1 TO U: Y=RND(100): GOSUB 5010: NEXT
320 Y=RND(70)+30: PRINT"*";: IF B(Y)>0 GOTO 320
330 B(Y)=5
340 Y=RND(100): IF B(Y)>0 GOTO 340
350 B(Y)=21: FOR X=1 TO 3
360 Y=RND(100): IF B(Y)>0 GOTO 360
370 B(Y)=22: NEXT
380 Y=RND(50): IF B(Y)>0 GOTO 380
390 B(Y)=23
400 Y=RND(50): IF B(Y)>0 GOTO 400
410 B(Y)=24
420 Y=RND(90): IF B(Y)>0 GOTO 420
430 B(Y)=25
440 Y=RND(80)+15: IF B(Y)>0 GOTO 440
450 B(Y)=26: FOR X=1 TO 5
460 Y=RND(80)+15: IF B(Y)>0 GOTO 460
470 B(Y)=27: NEXT
480 Y=RND(100): IF B(Y)>0 GOTO 480
490 B(Y)=26
500 Z=6: FOR X=1 TO 10: Y=RND(80)+20: GOSUB 5010: NEXT
510 Z=8: FOR X=1 TO 10: Y=RND(100): GOSUB 5010: NEXT
520 Y=RND(100): IF B(Y)>0 GOTO 520
530 B(Y)=7: Z=10: FOR X=1 TO 10: Y=RND(95)+5: GOSUB 5010:
    NEXT
540 Y=RND(100): IF B(Y)>0 GOTO 540
550 B(Y)=11: QQ=0
560 F=RND(100): IF B(F)>0 GOTO 560
570 B(F)=15
580 Y=RND(55)+45: IF B(Y)>0 GOTO 580
590 B(Y)=17: BZ=0
599 REM * INTRO *
600 CLS: PRINT: PRINT"  ","GREETINGS,  ";N$: PRINT
610 PRINT"You have joined the Magical creatures of the Woodlands in their"
620 PRINT"Glorious Quest to recover the Golden Flute of Love from the"
```

```
630 PRINT"clutches of the evil gremlin, Terak, and to return it to the"
640 PRINT"Woodlands. At all times watch out for dragons, Sirens,"
650 PRINT"goblins, gargoyles, and the dreaded Hopeless Pits!"; PRINT
660 PRINT"Coramble, the Great Oracle of Purlicon Mountain may be of"
670 PRINT"assistance.": PRINT
680 INPUT"Do you want directions";Q$: Q$ = LEFT$(Q$,1)
690 IF Q$ = "Y" GOSUB 5030
695 LC = 1: A(LC) = 12
699 REM * CHARIOT CONTENTS *
700 PRINT: GOSUB 5000
710 PRINT"  ","The Magic Chariot is now carrying—"
720 PRINT N$;"  the Human"
730 IF BR = 1 PRINT"BROMBIRAN the Elf"
740 IF DG = 1 PRINT"DAGGLETTE the Elf"
750 IF GR = 1 PRINT"GROMPHLUR the Elf"
760 IF RU = 1 PRINT"RULF the Satyr"
770 IF SJ = 1 PRINT"SEJJAN the Fairy"
780 IF JS = 1 PRINT"JESSAN the Fairy"
790 IF AL = 1 PRINT"ALLEGRECIA the Queen of the Sprites"
800 IF PM = 1 PRINT"PRINCESS MELVA"
810 IF FL = 1 PRINT"The Golden Flute"
820 IF MO = 0 PRINT"A Magic Orb"
830 IF SW = 1 PRINT"A Magic Sword"
840 IF BZ = 1 PRINT"A Magic Bazooka"
850 IF GL < 1 GOTO 880
860 PRINT GL;"piece";: IF GL > 1 PRINT"s";
870 PRINT"  of gold"
880 IF MZ = 1 PRINT"A Magic Zither"
890 INPUT"Please press 'ENTER'  ";Q$
899 REM * MAIN PLAY *
900 ZZ = LC: PRINT"You are at location  ";LC,"Move #";M
910 INPUT"Your move? (Enter 'K' for key)  ";Q$: Q$ = LEFT$(Q$,1)
920 IF Q$ = "K" GOTO 4000
925 IF Q$ = "M" GOTO 7000
930 IF Q$ = "B" AND BZ = 1 GOTO 4030
935 M = M + 1: A(LC) = B(LC)
940 IF Q$ = "D" GOTO 1000
950 IF Q$ = "U" GOTO 1190
960 IF Q$ = "R" GOTO 1200
970 IF Q$ = "L" GOTO 1220
980 PRINT"  ","INVALID MOVE!"
990 GOTO 910
1000 LC = LC + 10: IF LC > 100 GOTO 1020
1010 GOTO 4250
```

```
1020 PRINT: PRINT"The Magic Chariot has left the boundaries of the Magic
     Kingdom": PRINT
1030 GOSUB 5000
1040 PRINT"Without Magic to hold it up, the Magic Chariot crashes!": PRINT
1050 GOSUB 5000: M = M - 1: PRINT"You lose this time."
1060 PRINT"IT TOOK YOU  ";M;"  MOVES."
1070 IF PW = 0 OR PW = M GOTO 1100
1080 PRINT"Previous win record was  ";PW
1100 PRINT
1110 GW = GW + 1: PRINT"This was game #";GW
1120 PRINT"  ",: INPUT"PLAY AGAIN";Q$: Q$ = LEFT$(Q$,1)
1130 IF Q$ = "Y" GOTO 8000
1140 END
1150 M = M - 1: PRINT"You win this time": PRINT
1155 PRINT"It took you  ";M;"  moves!"
1160 IF PW = 0 GOTO 1175
1170 PRINT"PREVIOUS BEST SCORE WAS  ";PW: IF PW < M GOTO 1100
1175 PW = M: GOTO 1100
1190 LC = LC - 10: IF LC < 1 GOTO 1020 ELSE GOTO 4250
1200 LC = LC + 1: X = LC/10: Y = (X - INT(X))*10: IF(Y = 0) OR (Y > 1.5)
     GOTO 4250
1210 GOTO 1020
1220 LC = LC - 1: X = LC/10: Y = (X - INT(X))*10: IF Y = 0 GOTO 1020
1230 GOTO 4250
4000 PRINT"U = Up, D = Down, R = Right, L = Left, M = Map,  ";
4010 IF BZ > 0 PRINT"B = Fire Bazooka,  "
4020 PRINT"K = Key": GOTO 910
4030 INPUT "Bazooka loaded - Aim? (Enter 'X' to disarm)  ";Q$
4040 Q$ = LEFT$(Q$,1): IF Q$ = "X" GOTO 900
4050 IF Q$ = "U" GOTO 4100
4060 IF Q$ = "D" GOTO 4160
4070 IF Q$ = "R" GOTO 4190
4080 IF Q$ = "L" GOTO 4210
4090 GOTO 4030
4100 BU = 10000: BS = 1: BT = - 10
4110 PRINT"  ","KER-";: FOR X = 1 TO 55: NEXT: PRINT"POW!!": PRINT:
     X = LC
4120 X = X + BT: IF X < BS GOTO 900
4130 IF X > BU GOTO 900
4140 IF (X < 1) OR (X > 100) GOTO 900
4150 A(X) = 20: B(X) = 20: GOTO 4120
4160 BU = 10000: BS = 0: BT = 10
4170 GOTO 4110
4180 PRINT"  ","phffft!": PRINT: GOTO 900
```

```
4190  G = INT(LC/10): IF G = LC/10 GOTO 4180
4200  BS = 0: BU = G*10+10: BT = 1: GOTO 4110
4210  G = INT(LC/10): H = LC/10 - G: IF H = 0.1 GOTO 4180
4220  BU = 10000: BS = G*10+1: BT = -1: GOTO 4110
4250  PRINT: PRINT: PRINT: A(LC) = 12
4260  XX = B(LC)
4270  IF(LC = 1) OR (LC = 2) OR (LC = 11) OR (LC = 12) GOTO 10000
4280  IF LC = 33 GOSUB 5120
4290  IF(LC > 21 AND LC < 25) OR (LC > 41 AND LC < 45) GOSUB 5300
4300  IF LC = 32 OR LC = 34 GOSUB 5300
4310  IF XX = 4 GOSUB 5310
4320  P = RND(150): IF (P > 148) AND (FL = 0) GOTO 10030
4330  IF XX = 3 GOTO 10080
4340  IF XX = 5 GOTO 10200
4350  X = LC - 1: GOSUB 5580
4360  X = LC+1: GOSUB 5580
4370  X = LC - 10: GOSUB 5580
4380  X = X - 1: GOSUB 5580
4390  X = X+2: GOSUB 5580
4400  X = LC+10: GOSUB 5580
4410  X - 1: GOSUB 5580
4420  X+2: GOSUB 5580
4430  IF XX = 6 GOTO 10240
4440  IF XX = 7 GOSUB 5600
4450  IF XX = 8 GOSUB 5610
4460  IF XX = 9 GOTO 10250
4470  IF XX = 10 GOTO 10260
4480  IF XX = 11 GOSUB 5630
4490  IF XX = 13 OR XX = 14 GOTO 11700
4500  IF XX = 15 GOTO 11780
4510  IF XX = 17 GOSUB 5640
4520  IF XX = 20 PRINT"This area is a smouldering ruin."
4525  IF XX = 21 GOTO 10270
4527  IF XX = 22 GOTO 10280
4530  IF XX = 23 GOSUB 5660
4540  IF XX = 24 GOSUB 5670
4550  IF XX = 25 GOSUB 5750
4560  IF XX = 26 GOSUB 6000
4570  IF XX = 27 GOTO 11800
4580  IF XX = 28 GOSUB 6010
4590  IF XX = 29 GOSUB 6050
4900  GOTO 700
4999  STOP
5000  FOR TT = 1 TO 234: NEXT: RETURN
5009  REM * BLANK SPACE CHECK *
```

```
5010  IF B(Y) = 0 THEN B(Y) = Z
5020  PRINT"#";: RETURN
5029  REM * INSTRUCTIONS *
5030  PRINT"Each move may be Up, Down, Right, or Left. Only the first"
5040  PRINT"letter is needed. Never leave the map boundaries!"
5050  PRINT"If you want to see a map of the Magic Kingdom, enter 'M' as"
5060  PRINT"your move. This will not increment your move counter."
5070  PRINT"If you have the Magic Bazooka, you may enter 'B' to fire."
5080  PRINT"To have Terak's location displayed on the map, you must visit"
5090  PRINT"the Oracle on Purlicon mountain. If you are carrying a Magic"
5100  PRINT"Orb he may even tell you more. Or he may not."
5110  PRINT"The object of the game is to return the Golden Flute to the"
5115  PRINT"Woodlands in the fewest moves. Please press 'ENTER'  ";: INPUT
      Q$: RETURN
5120  IF XX = 20 GOTO 5290
5130  PRINT"Coramble, the Great Oracle, reveals that Terak's lair is"
5140  PRINT"at location #";
5150  FOR Y = 1 TO 100: IF B(Y) = 4 GOTO 5170
5160  NEXT: IF MO = 1 GOTO 5180 ELSE RETURN
5170  PRINT Y: A(Y) = 4: GOTO 5160
5180  IF AX = 3 THEN RETURN
5190  AX = AX+1: A = RND(6)+4: IF(A = 5) OR (A = 6) OR (A = 9) OR
      (A = 10) GOTO 5210
5200  RETURN
5210  PRINT"He also reveals the location of all  ";
5220  IF A = 5 PRINT"Sirens."
5230  IF A = 6 PRINT"dragons."
5240  IF A = 9 PRINT"gargoyles."
5250  IF A = 10 PRINT"goblins."
5260  PRINT"They will be shown on your next map."
5270  FOR X = 1 TO 100: IF B(X) = A THEN A(X) = B(X)
5280  NEXT: RETURN
5290  PRINT"The Great Oracle's body lies smoking in the corner.": RETURN
5300  PRINT"You are at the foot of Purlicon Mountain.": RETURN
5310  PRINT"You have infiltrated Terak's lair!": GOSUB 5000
5320  IF FL = 1 GOTO 5360
5330  FL = 1: PRINT"You recover the Golden Flute!"
5340  IF MO > 0 GOTO 5540
5350  RETURN
5360  D = RND(8): IF D = 1 AND BR = 0 GOTO 5530
5370  IF D = 2 AND DG = 0 GOTO 5530
5380  IF D = 3 AND GR = 0 GOTO 5530
5390  IF D = 4 AND RU = 0 GOTO 5530
5400  IF D = 5 AND SJ = 0 GOTO 5530
5410  IF D = 6 AND JS = 0 GOTO 5530
```

```
5420  IF D = 7 AND AL = 0 GOTO 5530
5430  IF D = 8 AND PM = 0 GOTO 5530
5440  IF D = 1 PRINT"Brombiran";: BR = 0
5450  IF D = 2 PRINT"Dagglette";: DG = 0
5460  IF D = 3 PRINT"Gromphlur";: GR = 0
5470  IF D = 4 PRINT"Rulf";: RU = 0
5480  IF D = 5 PRINT"Sejjan:";: SJ = 0
5490  IF D = 6 PRINT"Jessan";: JS = 0
5500  IF D = 7 PRINT"Allegrecia";: AL = 0
5510  IF D = 8 PRINT"Princess Melva";: PM = 0
5520  PRINT"  is dead.": PRINT
5530  RETURN
5540  A = RND(3): IF A = 1 GOTO 5360
5550  IF A = 3 GOTO 5570
5560  RETURN
5570  PRINT"The Magic Orb is destroyed!": MO = 0: RETURN
5580  IF (X < 1) OR (X > 100) THEN RETURN
5590  IF B(X) = 5 PRINT"An eerie sweet singing is heard in the distance."
5595  RETURN
5600  PRINT: PRINT"You just found a Magic Sword!": B(LC) = 0: SW = 1:
      RETURN
5610  X = RND(199) + 1: PRINT"You just found  ";X;"  pieces of gold!"
5620  B(LC) = 0: GL = GL + X: PRINT: RETURN
5630  PRINT"You just found a Magic Orb!": PRINT: MO = 1: B(LC) = 0:
      RETURN
5640  B(LC) = 0: PRINT"You just found a  ";: GOSUB 5000
5650  PRINT"***** MAGIC BAZOOKA *****": BZ = 1: RETURN
5660  G$ = "KLUFFOOT": GOTO 5680
5670  G$ = "FRIEK"
5680  IF FL = 0 THEN RETURN
5690  PRINT G$;", THE GARGOYLE SLAVE OF TERAK,"
5700  IF SW = 1 GOTO 5720
5710  PRINT" STEALS THE GOLDEN FLUTE AGAIN!": FL = 0: RETURN
5720  PRINT" ATTEMPTS TO STEAL THE GOLDEN FLUTE AGAIN!": PRINT
5730  PRINT"BUT THE MAGIC SWORD KILLS  ";G$;"!": PRINT
5740  B(LC) = 0: RETURN
5750  PRINT"You are in the Enchanted Forest!": PRINT
5760  ZX = BR + DG + GR + RU + SJ + JS + AL + PM
5770  IF ZX > 5 THEN RETURN
5780  R = RND(10): IF R > 8 THEN RETURN
5790  IF (R = 1) AND (BR = 0) GOTO 5880
5800  IF (R = 2) AND (DG = 0) GOTO 5890
5810  IF (R = 3) AND (GR = 0) GOTO 5900
5820  IF (R = 4) AND (RU = 0) GOTO 5910
5830  IF (R = 5) AND (SJ = 0) GOTO 5920
```

```
5840 IF (R = 5) AND (JS = 0) GOTO 5930
5850 IF (R = 7) AND (AL = 0) GOTO 5940
5860 IF (R = 8) AND (PM = 0) GOTO 5950
5870 RETURN
5880 PRINT"BROMBIRAN";: BR = 1: GOTO 5970
5890 PRINT"DAGGLETTE";: DG = 1: GOTO 5970
5900 PRINT"GROMPHLUR";: GR = 1: GOTO 5970
5910 PRINT"RULF";: RU = 1: GOTO 5970
5920 PRINT"SEJJAN";: SJ = 1: GOTO 5970
5930 PRINT"JESSAN";: JS = 1: GOTO 5970
5940 PRINT"ALLEGRECIA";: AL = 1: GOTO 5970
5950 PRINT"PRINCESS MELVA";: PM = 1
5970 C(R) = 1.5: B(LC) = 0: PRINT"  IS MAGICALLY RESTORED TO LIFE!"
5980 PRINT: RETURN
6000 PRINT"You just found a Magic Zither!": PRINT: MZ = 1: B(LC) = 0:
     RETURN
6010 PRINT"THE MAGIC CHARIOT JUST FLEW OVER AN ENCHANTED
     LAND MINE!": PRINT
6020 GOSUB 5000: FOR X = 1 TO 50: XZ = RND(1022): PRINT@XZ,"*";
6030 U = XZ*X: NEXT: PRINT: PRINT
6040 GOTO 5360
6050 IF GL < 2 THEN RETURN
6060 PRINT"A wicked witch steals all of your gold coins!": PRINT
6070 GL = 0: PRINT: PRINT"  ","Hee hee hee!": PRINT: PRINT
6080 RETURN
6999 REM * MAP *
7000 CLS: PRINT: PRINT: Z = 1: FOR X = 1 TO 10: PRINT"  ",
7010 FOR Y = 1 TO 10: V = A(Z)
7020 IF(V = 0) OR (V = 7) OR (V = 8) OR (V = 11) OR (V > 15 AND V < 20) OR
     (V > 22 AND V < 27) PRINT".  ";
7030 IF V = 1 PRINT"W  ";
7040 IF V = 2 PRINT"M  ";
7050 IF V = 3 PRINT"P  ";
7060 IF V = 4 PRINT"T  ";
7070 IF V = 5 PRINT"S  ";
7080 IF V = 6 PRINT"D  ";
7090 IF V = 9 PRINT"G  ";
7100 IF V = 10 PRINT"g  ";
7110 IF V = 12 PRINT"C  ";
7120 IF (V = 13) OR (V = 14) PRINT"X  ";
7130 IF V = 15 PRINT"F  ";
7140 IF V = 20 PRINT"*  ";
7150 IF V = 21 PRINT"s  ";
7160 IF V = 22 PRINT"d  ";
7170 IF V = 27 PRINT"R  ";
```

```
7180  IF V = 28 PRINT"!  ";
7190  IF V = 29 PRINT"w  ";
7200  Z = Z + 1: NEXT Y: PRINT: NEXT X
7210  PRINT"C  The Magic Chariot (YOU), D = Dragon, d = Dwarf, F = Fog,"
7220  PRINT"G = Gargoyle, g = goblin, M = Purlicon Mountain, P = The
      Hopeless"
7230  PRINT"Pits, R = Rock, S = Sirens, s = Magic Sparrow, T = Terak's lair,"
7240  PRINT"W = The Woodlands, w = Witch, ! = Land mine, * = smouldering
      ruin"
7250  GOTO 900
8000  FOR X = 1 TO 100: A(X) = 0: B(X) = 0: PRINT"&";: NEXT
8010  GOTO 70
1000  PRINT"YOU ARE IN THE WOODLANDS, HOME OF THE ELVES.":
      PRINT
10010 IF FL = 1 PRINT"YOU HAVE RECOVERED THE GOLDEN FLUTE!":
      GOTO 1150
10020 GOTO 4280
10030 PRINT"Terak fears your approach and moves his lair!": PRINT
10040 FOR X = 1 TO 100: IF B(X) = 4 THEN B(X) = 0: A(X) = 0
10050 NEXT
10060 X = RND(100): IF B(X) > 1 GOTO 10060
10070 B(X) = 4: GOTO 4310
10080 PRINT"The Magic Chariot is mired in the Hopeless Pits!": PRINT
10090 IF MO = 1 GOTO 10130
10100 A = RND(5): IF A > 3 PRINT"YOU ARE DOOMED!": GOTO 1050
10110 IF A < 3 GOSUB 5360
10120 PRINT"You manage to get the Magic Chariot free of the muck.": GOTO
      700
10130 INPUT"Do you rub your Magic Orb";Q$: Q$ = LEFT$(Q$,1)
10140 IF Q$ = "Y" GOTO 10160
10150 GOTO 10100
10160 GOSUB 5000: CLS: PRINT: PRINT: PRINT: PRINT
10170 PRINT"  ","*** POOF ***": PRINT: PRINT: PRINT
10180 LC = RND(100): PRINT"The Magic Chariot is magically transported to
      location #  ";LC
10190 A(ZZ) = B(ZZ): A(LC) = 12: GOTO 700
10200 PRINT: PRINT"The Sweet Song of the Sirens mesmerizes you!": PRINT
10210 GOSUB 5000
10220 PRINT"The Magic Chariot crashes!": PRINT: GOSUB 5000
10230 GOTO 1050
10240 PRINT"A dragon  ";: FO = 1: GOTO 10300
10250 PRINT"A gargoyle  ";: FO = 2:GOTO 10300
10260 PRINT"A goblin  ";: FO = 3 GOTO 10300
10270 PRINT"A Magic Sparrow  ";: FO = 4: GOTO 10300
10280 PRINT"An ancient dwarf  ";: GM = RND(150): FO = 5
```

```
10300  AM = RND(4): PRINT"is in your path!": PRINT: GOSUB 5000
10310  A(LC) = B(LC)
10320  PRINT"  ","POSSIBLE ACTIONS": PRINT
10330  PRINT"1 — Move the Magic Chariot"
10340  PRINT"2 — Throw some gold coins overboard"
10350  PRINT"3 — Hand to hand combat"
10360  IF MO = 1 PRINT"4 — Rub Magic Orb"
10370  IF SW = 1 PRINT"5 — Unsheath Magic Sword"
10380  IF MZ = 1 PRINT"6 — Play Magic Zither"
10390  PRINT"  ",: INPUT"YOUR CHOICE";P
10400  IF P = 1 GOTO 10490
10410  IF P = 2 GOTO 10660
10420  IF P = 3 GOTO 11000
10430  IF(MO = 1) AND (P = 4) GOTO 10160
10440  IF (SW = 1) AND (P = 5) GOTO 11500
10450  IF (MZ = 1) AND (P = 6) GOTO 11630
10460  PRINT"INVALID SELECTION!": PRINT: GOTO 10320
10490  IF FO > 3 GOTO 910
10500  PRINT: INPUT"DIRECTION";M$: M$ = LEFT$(M$,1)
10520  IF M$ = "U" GOTO 10570
10530  IF M$ = "D" GOTO 10580
10540  IF M$ = "R" GOTO 10590
10550  IF M$ = "L" GOTO 10600
10560  GOTO 10500
10570  IF AM = 1 GOTO 1190 ELSE GOTO 10610
10580  IF AM = 2 GOTO 1000 ELSE GOTO 10610
10590  IF AM = 3 GOTO 1200 ELSE GOTO 10610
10600  IF AM = 4 GOTO 1220
10610  PRINT: PRINT"  The  ";: IF FO = 1 PRINT"dragon";
10620  IF FO = 2 PRINT"gargoyle";
10630  IF FO = 3 PRINT"goblin";
10640  PRINT"  will not let you go that way!"
10650  GOTO 10320
10660  PRINT: INPUT"How much gold do you toss overboard";H
10670  A = ABS(H): H = INT(H): IF H = 0 GOTO 10660
10680  IF H > GL GOTO 10995
10690  GL = GL – H
10700  IF FO = 1 GOTO 10900
10710  IF FO = 2 GOTO 10910
10720  IF FO = 3 GOTO 10940
10730  IF FO = 4 GOTO 10980
10740  PRINT"The dwarf thanks you very politely  "
10750  GM = GM – H: IF GM > 1 GOTO 10320
10760  PRINT"and he reveals the location of all  ";
10770  DF = RND(4): IF DF = 1 GOTO 10840
```

```
10780  IF DF = 2 GOTO 10860
10790  IF DF = 3 GOTO 10880
10800  PRINT"Sirens": FOR X = 1 TO 100: IF B(X) = 5 THEN A(X) = 5
10810  NEXT
10820  PRINT"This information will be displayed on your next map."
10830  A(LC) = 12: GOTO 700
10840  PRINT"goblins": FOR X = 1 TO 100: IF B(X) = 10 THEN A(X) = 10
10850  NEXT: GOTO 10820
10860  PRINT"gargoyles": FOR X = 1 TO 100: IF B(X) = 9 THEN A(X) = 9
10870  NEXT: GOTO 10820
10880  PRINT"dragons": FOR X = 1 TO 100: IF B(X) = 6 THEN A(X) = 6
10890  NEXT: GOTO 10820
10900  PRINT"Dragons have no interest in gold.": PRINT: GOTO 10320
10910  PRINT"The gargoyle puts the gold into its sack  ";
10920  I = RND(200): IF H < I THEN PRINT: GOTO 10320
10930  PRINT"and leaves": B(LC) = 0: A(LC) = 12: GOTO 700
10940  I = RND(200): PRINT"The goblin eats the gold greedily  ";
10950  IF H < I THEN PRINT: GOTO 10320
10960  PRINT"and dies of": PRINT"terminal indigestion.": PRINT
10970  B(LC) = 0: A(LC) = 12: GOTO 700
10980  PRINT"What would a bird want with  ";H;"  pieces of gold?": PRINT
10990  GOTO 10320
10995 . . .PRINT"YOU DO NOT HAVE  ";H;"  PIECES OF GOLD!": GOSUB
5360: GOTO 10320
11000  PRINT: PRINT"  ","YOUR CHAMPION?"
11010  IF BR = 1 PRINT"1 – BROMBIRAN"
11020  IF DG = 1 PRINT"2 – DAGGLETTE"
11030  IF GR = 1 PRINT"3 – GROMPHLUR"
11040  IF RU = 1 PRINT"4 – RULF"
11050  IF SJ = 1 PRINT"5 – SEJJAN"
11060  IF JS = 1 PRINT"6 – JESSAN"
11070  IF AL = 1 PRINT"7 – ALLEGRECIA"
11080  IF PM = 1 PRINT"8 – PRINCESS MELVA"
11090  PRINT"9 –  ";N$
11100  PRINT"  ",: INPUT"YOUR CHOICE";CH
11110  IF CH = 1 AND BR = 1 GOTO 11210
11120  IF CH = 2 AND DG = 1 GOTO 11220
11130  IF CH = 3 AND GR = 1 GOTO 11230
11140  IF CH = 4 AND RU = 1 GOTO 11240
11150  IF CH = 5 AND SJ = 1 GOTO 11250
11160  IF CH = 6 AND JS = 1 GOTO 11260
11170  IF CH = 7 AND AL = 1 GOTO 11270
11180  IF CH = 8 AND PM = 1 GOTO 11280
11190  IF CH = 9 GOTO 11290
11200  PRINT"  ","WHO???": PRINT: GOTO 11010
11210  I = 50: GOTO 11300
```

```
11220  I=55: GOTO 11300
11230  I=45: GOTO 11300
11240  I=85: GOTO 11300
11250  I=25: GOTO 11300
11260  I=25: GOTO 11300
11270  I=50: GOTO 11300
11280  I=40: GOTO 11300
11290  I=75
11300  H=I*C(CH): C(CH)=C(CH)-.1: GOSUB 5000
11310  PRINT: PRINT: RS=RND(100)
11320  IF FO=4 GOTO 11390
11330  IF FO=5 GOTO 11490
11340  IF RS>H GOTO 11390
11350  PRINT"THE  ";: IF FO=1 PRINT"DRAGON";
11360  IF FO=2 PRINT"GARGOYLE";
11370  IF FO=3 PRINT"GOBLIN";
11380  PRINT"  IS SLAIN!": PRINT: B(LC)=0: A(LC)=12: GOTO 700
11390  IF CH=1 PRINT"BROMBIRAN";: BR=0
11400  IF CH=2 PRINT"DAGGLETTE";: DG=0
11410  IF CH=3 PRINT"GROMPHLUR";: GR=0
11420  IF CH=4 PRINT"RULF";: RU=0
11430  IF CH=5 PRINT"SEJJAN";: SJ=0
11440  IF CH=6 PRINT"JESSAN";: JS=0
11450  IF CH=7 PRINT"ALLEGRECIA";: AL=0
11460  IF CH=8 PRINT"PRINCESS MELVA";: PM=0
11470  IF CH=9 PRINT N$;"  IS SLAIN!": PRINT"GAME OVER!": GOTO 1050
11480  C(CH)=0: PRINT"  IS SLAIN!": PRINT: GOTO 10320
11490  PRINT"THE DWARF IS SLAIN!": B(LC)=0: A(LC)=12: GOTO 700
11500  Q=RND(100): IF FO=1 GOTO 11550
11510  IF FO=2 GOTO 11600
11520  IF FO=3 GOTO 11350
11530  IF FO=4 GOTO 11390
11540  IF FO=5 GOTO 11490
11550  IF Q<65 GOTO 11350
11560  PRINT"This dragon is immune to your sword!": PRINT: GOSUB 5000
11570  PRINT"It attacks the Magic Chariot!": PRINT: GOSUB 5000
11580  GOSUB 5360: IF Q<80 GOTO 10320
11590  PRINT"Your Magic Sword is destroyed!": PRINT: SW=0: GOTO 10320
11600  IF Q<50 GOTO 11350
11610  PRINT"The gargoyle draws its own Magic Sword!": PRINT: GOSUB 5000
11620  GOTO 11580
11630  IF FO=1 GOTO 11350
11640  IF FO=3 GOTO 11670
11650  PRINT"No one is particularly impressed with your talent.": PRINT
11660  GOTO 10320
```

```
11670  Q = RND(10): IF Q > 5 GOTO 11350
11680  GOSUB 5360: GOTO 10320
11699  REM * WALL *
11700  PRINT: PRINT"You just ran into a brick wall!": PRINT
11710  QQ = RND(3) + QQ: IF QQ < 7 GOSUB 5360
11720  IF QQ > 10 GOTO 11750
11730  A(LC) = B(LC): IF XX = 13 THEN LC = LC - 20 ELSE LC = LC + 20
11740  PRINT"THE MAGIC CHARIOT IS HURLED BACKWARDS!": GOTO
       4250
11750  PRINT"The Magic Chariot is reduced to a pile of junk.": PRINT
11760  GOSUB 5000
11770  PRINT"You and your entire party are dead.": PRINT: GOTO 1050
11780  PRINT"The Magic Chariot has flown into a thick, mysterious fog!": PRINT:
       PRINT
11790  FOR X = 1 TO 150: PRINT"#";: NEXT: A(LC) = 15: LC = RND(50) + 30
11795  PRINT"The fog clears.": ZZ = A(LC): A(LC) = 12: GOTO 700
11800  PRINT: PRINT"A gigantic boulder blocks your path!": PRINT
11810  R = RND(4)
11820  A(LC) = 27: ZZ = 27
11830  INPUT"Which way do you try to go";M$: M$ = LEFT$(M$,1)
11840  M = M + 1
11850  IF M$ = "D" AND R = 1 GOTO 1000
11860  IF M$ = "U" AND R = 2 GOTO 1190
11870  IF M$ = "R" AND R = 3 GOTO 1200
11880  IF M$ = "L" AND R = 4 GOTO 1220
11890  PRINT"THE BOULDER IS IN YOUR WAY!": PRINT
11895  FOR X = 1 TO 9: C(X) = C(X) - (C(X)/20): NEXT: GOTO 11830
```

THE PLAY

The playing area for THE GOLDEN FLUTE is similar to the one used in MARS. It is a 10 X 10 space map contained in arrays. Of course, this gives 100 possible places for the Magic Chariot to visit.

In MARS, if the player went past the boundaries of the map area, he looped around to the opposite end of the map. In THE GOLDEN FLUTE, a different approach is used. Leaving the legal boundaries of the defined playing area in this game results in an instant game loss. This is programmed in lines 1000, 1020 through 1050, and 1190 through 1220. If the player ever selects an out of bounds move, he forfeits the game, and the compuer prints out the following message:

> THE MAGIC CHARIOT HAS LEFT THE BOUNDARIES OF THE MAGIC KINGDOM. WITHOUT MAGIC TO HOLD IT UP, THE MAGIC CHARIOT CRASHES!

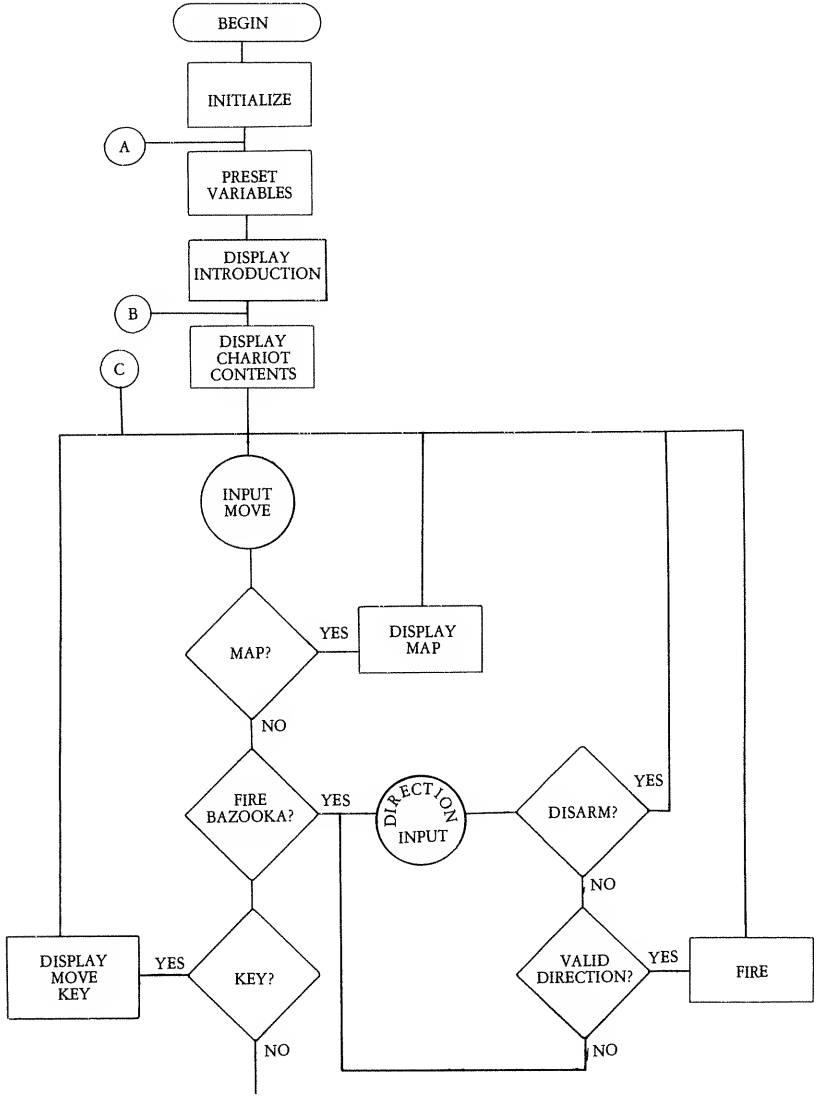


Figure 9.1A Flow Chart for THE GOLDEN FLUTE Program.

As in MARS, two maps of the playing area are set up in THE GOLDEN FLUTE program with two arrays--A(x) and B(x). This technique is extremely useful in a great many different adventure games.

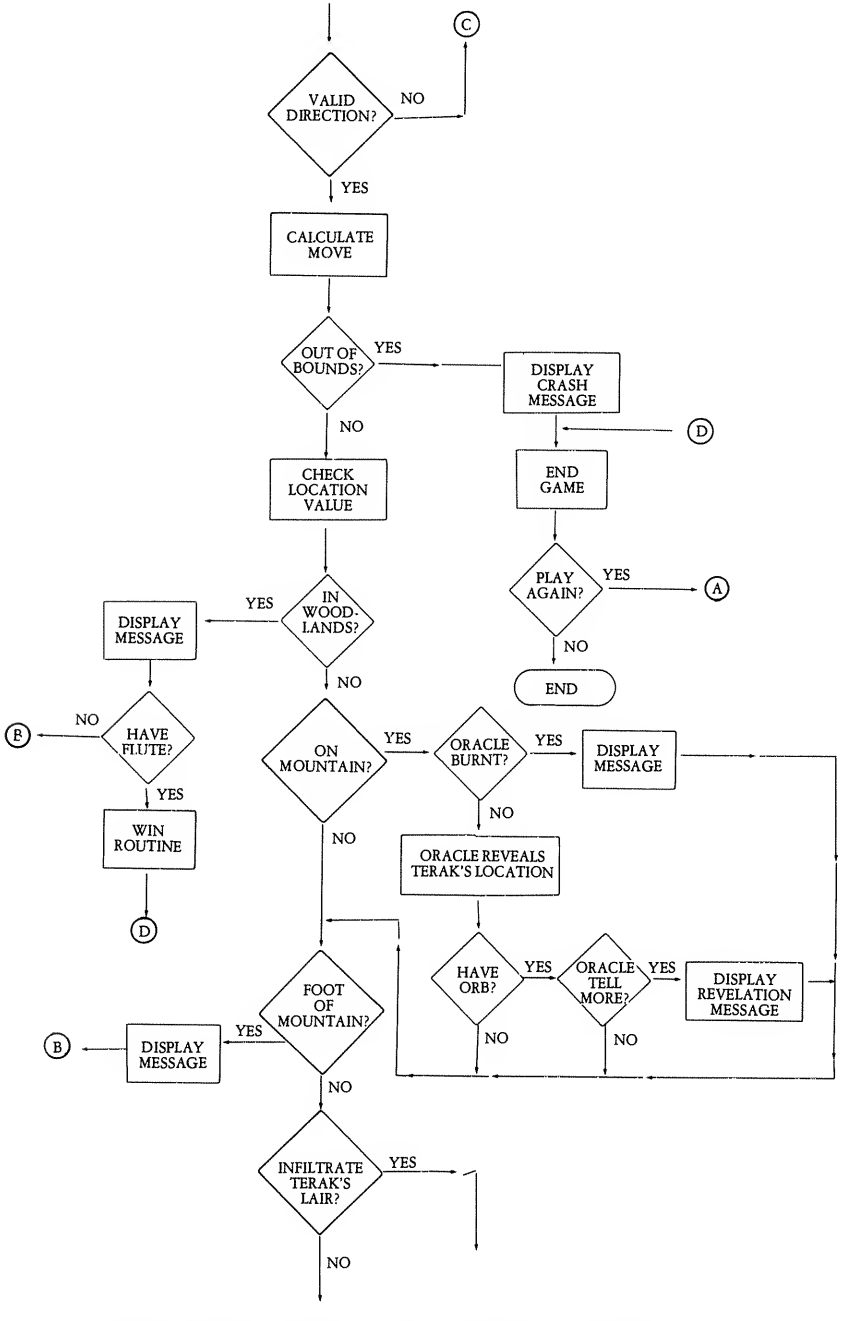


Figure 9.1B Flow Chart for THE GOLDEN FLUTE Program.

Array A(x) contains information about explored and fixed areas of the Magic Kingdom. This is the map that is displayed in lines 7000 through 7250 of the program. Table 9.5 lists the various values that may be placed at each map location. Numbers that are not listed (for example, 16 or 18) are not used in the program as it is given here. You can use these unassigned values to set up your own additions to the game.

Table 9.5 also shows the character used to indicate each item on the map. You'll notice that several items are marked only with a period (".). These are items that are invisible on the displayed map. It is up to the player to remember where these things are. Some items such as the Magic Sword and pieces of gold have no character symbol at all. These items are automatically loaded onto the Magic Chariot as soon as they are found, so the space is left empty. There is nothing to display. The space is treated like any blank location.

Array B(x), on the other hand, is a complete map of the playing area. This is the map used by the computer whenever the player makes a move. The **IF...THEN...** statements in lines 4270 through 4900 are used to branch the program to the appropriate subroutine.

Throughout the game, a number of creatures may appear in the path of the Magic Chariot. These include ancient dwarfs, dragons, gargoyles, goblins, and magic sparrows. Each time one of these creatures is encountered, the computer displays a standardized list of possible actions for the player to choose from. Lines 10320 through 10380 display this list. If an invalid selection is made, the computer simply loops back around and asks for a new response.

The possible actions when facing one of these creatures include:

1—MOVE CHARIOT
2—THROW SOME GOLD OVERBOARD
3—HAND TO HAND COMBAT
4—RUB MAGIC ORB
5—UNSEATH MAGIC SWORD
6—PLAY MAGIC ZITHER

The first three choices (move chariot, throw gold, or combat) are always offered. The other three are only printed on the menu and accepted by the program if the appropriate object (Magic Orb, Magic Sword, or Magic Zither) is aboard the Magic Chariot.

Any of these actions will be useful in some circumstances. At other times they may work against the player's best interests. None of these actions will be a good choice in all cases.

Part of the appeal of adventure games is figuring out how to out maneuver the creatures so the book won't help you here. However, a good

adventure game is still fun to play, even after you've solved all the puzzles. Plenty of **RND** (random) statements in the program keep a game from going stale. But the player should use logic to deal with all obstacles, not just blind luck.

In any case, consider here a few details of each of the actions you could choose from when facing a creature in THE GOLDEN FLUTE.

If the player elects to move the Magic Chariot, he is queried for his desired direction. Sometimes a creature will allow the chariot to go one way but will block moves in all other directions. At other times, it will block all four directions. A few creatures don't care where the Chariot goes.

Each attempted move increments the move counter, adding to the player's score, even if the move is blocked off. Remember, THE GOLDEN FLUTE is scored like golf, with the object being to complete the game with as few moves as possible.

The second possible action is to throw pieces of gold overboard. Some creatures have no interest at all in gold. (A little logic should suggest which ones don't care.) Each of the other creatures determines how much gold it wants (a **RND** statement is used). If the player throws less than this amount of gold overboard, it will not help him. If the player equals or exceeds the creature's price, it is to the player's advantage. However, always bear in mind that it can be extremely dangerous to throw away more gold than you have.

The third possible action is hand to hand combat. When a player selects this alternative, the player is asked to pick a champion from among the characters aboard the Magic Chariot. He may not use a dead character.

Each character has a different strength rating. For example, the satyr is much stronger than either of the fairies. However, the monster (which has a randomly selected strength rating) is weakened by each battle. Jessan, the fairy, could conceivably slay a dragon that has just killed Rulf, the satyr.

The human (the player's character) is given the highest strength rating. But he should avoid going into hand to hand combat himself, unless everyone else in the Questing Party is already dead. If the player's character is killed, the game ends immediately.

Unsheathing the Magic Sword is generally pretty effective in dealing with creatures blocking the path of the Magic Chariot. Of course, the player must have already found the Magic Sword to use it.

Some gargoyles may have their own Magic Swords. A few dragons are immune to the Sword's magic. When the player unsheathes his Magic Sword before one of these creatures, he runs the risk of losing a Quester, and/or the Sword itself.

The Magic Orb can be used to get out of tough spots, but it can place the player into an even tougher spot.

Playing the Magic Zither usually doesn't accomplish much; but in certain circumstances, it proves worthwhile.

Occasionally, the Magic Chariot's path will be blocked by a giant boulder. Only one of the four directions may be used to leave the location. The player cannot always backtrack. If the player runs into a boulder by moving up, it may be in his way when he tries to move down. It is always possible to move in one direction. Unsuccessful move attempts are added to the move counter.

The computer randomly selects the direction for passing a boulder each time the Chariot lands in a particular location. It can change if the player encounters the same boulder more than once in the same game. The boulders are as magical as everything else in this game.

If the player is on the edge of the playing area, a boulder can force him to move out of bounds, and he will have to forfeit the game. This could be very frustrating; but, fortunately, odds are that it will not happen often. Complicated and tricky programming would prevent this, but it may not be worth bothering with. Avoiding the perimeters of the Magic Kingdom is part of the game strategy.

To land on the location occupied by the Sirens is instantly fatal. Whenever the Magic Chariot lands on a space adjacent to the Sirens (including diagonals), a warning message is displayed.

Hidden somewhere within the Magic Kingdom is a dandy weapon called the Magic Bazooka. The relevant programming is in lines 4030 through 4220 and 5640 through 5650. Once the player has found the Magic Bazooka, he can fire in any direction he chooses, instead of making a regular move. The Magic Bazooka cannot be used at close range, so it cannot be used when facing a monster.

When fired, the Magic Bazooka destroys everything in its path, leaving only smoldering ruins. It can destroy Coramble, the oracle; Terak; the Golden Flute, if he has it; the Woodlands; and any pieces of gold or hidden weapons. The Magic Bazooka must be used with considerable discretion.

Table 9.2 Routines and Subroutines Used in the "Golden Flute" Program.

Routines	
10-30	initialize
40-100	main preset
110-220	preset known map
230-590	preset hidden items
600-695	display introduction

700-890	display contents of the Magic Chariot
900-990	main play routine
1000-1010	move Down
1020-1050	out of bounds
1060-1140	end game/new game
1150-1175	win game
1190	move Up
1200-1210	move Right
1220-1230	move Left
4000-4020	display possible move key
4030-4220	fire Magic Bazooka
4250-4900	current location check

Subroutines

5000	timing delay loop
5010-5020	check for blank location
5030-5115	instructions
5120-5290	visit Oracle
5300	foot of mountain display
5310-5350/5540-5570	infiltrate Terak's lair
5360-5530	character death
5580	check for out of bounds
5590-5595	"eerie singing in the distance" display
5600	find Magic Sword
5610-5620	find gold
5640-5650	find Magic Bazooka
5660-5740	Kluffoot/Friek attempts to re-steal Flute
5750-5980	Enchanted Forest/dead character revived
6000	find Magic Zither
6010-6040	enchanted land mine
6050-6080	wicked witch steals gold
7000-7250	display known map
8000-8010	clear maps
10000-10020	in the Woodlands message
10030-10070	Terak moves his lair
10080-10120	Magic Chariot mired in the Hopeless Pits
10130-10910	rub Magic Orb
10200-10230	Song of the Sirens
10240-10460	face monster
10490-10650	attempt to move past monster
10660-10995	toss gold overboard
11000-11490	hand to hand combat
11500-11620	unsheath Magic Sword
11630-11680	play Magic Zither

11700-11770	run into wall
11780-11795	mysterious fog
11800-11895	boulder

Table 9.3 Characters in the "Golden Flute" Game.

Aboard the Magic Chariot

N$ "the Human" (player)
"Brombiran, the Elf"
"Dagglette, the Elf"
"Gromphlur, the Elf"
"Rulf, the satyr"
"Jessan, the Fairy"
"Sejjan, the Fairy"
"Allegrecia, Queen of the Sprites"
"Princess Melva"

Potential Helpers

Coramble, the Great Oracle
dwarves

Villains

Terak (goal—has the Golden Flute)
Kluffoot
Friek
gargoyles
goblins
dragons
wicket witches
Magic sparrows

PLAYING THE GAME

It's a good idea to start the game with a visit to Purlicon Mountain. Coramble, the Great Oracle of the Mountain, will reveal Terak's location (and possibly other useful information).

Occasionally Terak will fear the approach of the Magic Chariot and rehide his lair. When this happens, the player may have to return to the Oracle to find out the gremlin's new location.

If the player has the Magic Orb, Coramble may also reveal the location of some Monsters.Then again, the ornery old cuss may not. This routine is programmed in lines 5120 through 5290.

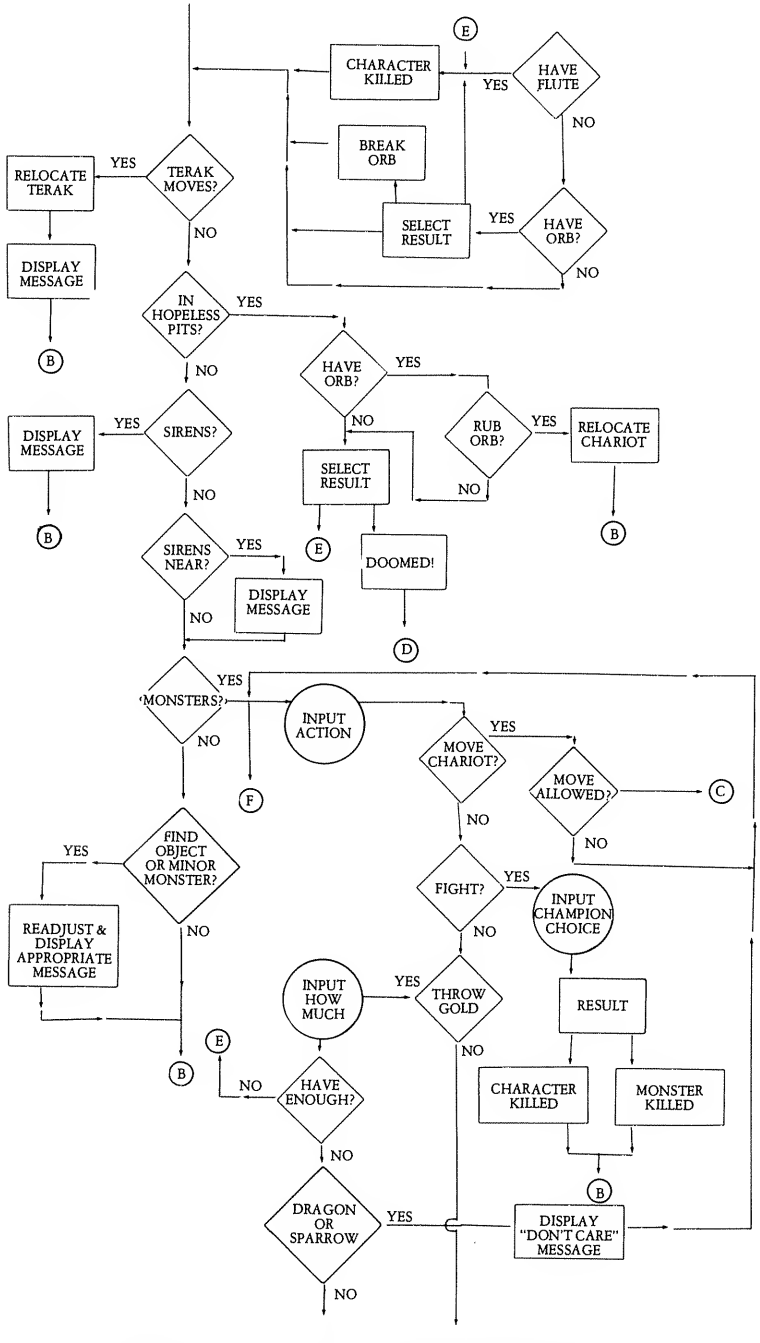


Figure 9.1C Flow Chart for THE GOLDEN FLUTE Program.

SOME SECRETS OF THE GAME

You may want to enter the program and play the game a few times before reading the rest of this chapter. Solving the various problems you encounter is part of the fun of many adventure games. What should you do when faced by a gargoyle? What if a Magic Sparrow is in your path? The discussion in the next few pages reveals some solutions.

As Table 9.2 shows, the basic game is played in lines 700 through 1230. This section accepts the player's move and checks it for validity. There are six legal entries. They are "K", "U", "D", "R", "L", and "B". Only the first letter of the player's response is used by the program.

Entering "K" will print out a key to the legal moves.

Generally, the player responds with one of the four directional moves. If you enter "U", the Magic Chariot moves up one space in the map. Similarly, an entry of "D" moves the Chariot down one space. "R" and "L" are used to move right and left. Remember that the player cannot move outside the boundaries of the map in any direction. No loop around is used in this game program.

If the player has located the Magic Bazooka, he has an additional possible choice. By entering "B", he can fire the Magic Bazooka. See program lines 4030 through 4220. When "B" is entered, the computer will prompt the player for the desired direction. (He may change his mind and disarm the Magic Bazooka). The Bazooka then reduces everything in the selected direction to a smoldering ruin.

Once the player enters a directional move, the program checks the hidden complete map (array B(x)) for monsters, obstacles, weapons, or gold coins. These checks are performed in lines 4250 through 4900.

After each turn, the computer will ask the player if he wants a display map of the Magic Kingdon (lines 7000 through 7250). The program could easily be adapted to draw a map after each turn, but this slows the game down and becomes monotonous. It takes several seconds for the map to be displayed, and it can be irritating to wait when you don't particularly need it.

The characters of Table 9.5 are used in the display map. Unexplored locations are displayed as blanks.

THE MONSTERS

Since the creatures that confront the Magic Chariot provide much of the excitement of the game, it is worthwhile to note how they are dealt with in the program. The basic procedure is essentially the same for all five types of creatures—dragons, dwarfs, gargoyles, goblins, and Magic Sparrows. Now examine what happens when the Magic Chariot encounters a goblin.

Goblins are stored in the map arrays with the value 10. When the current location (B(LC)) equals 10 the program jumps from line 4470 to line 10260.

The first few steps at 10260 print out the type of monster (A GOBLIN), and the foe variable (FO) is set to a value of 3. This variable is used so that all the monsters can use the same choice of action routine (lines 10300 through 10460), yet separate results may be easily obtained for each type of creature.

The possible actions offered were discussed earlier. Now we will examine what happens with each option when the Magic Chariot is facing a goblin.

1. MOVE CHARIOT. In line 10300 a random number from 1 to 4 is generated. This number is assigned to the variable AM, and indicates the only move direction allowed. If AM equals 1, the Magic Chariot may only move up, and so forth.

When option 1 has been selected, the player is queried for his choice of direction (line 10500). The move counter is incremented, and the player's choice is compared to the value of AM. If the player has made the correct entry, the Chariot moves in the ordinary manner; and the monster will be displayed on future display maps. If, on the other hand, an unallowed move is made, the goblin will block the path of the Magic Chariot (lines 10610 through 10650), and the program jumps back to line 10320 to ask for another choice of action. Remember, the object of the game—return the Flute to the Woodlands in as few moves as possible. A move blocked by a monster is still counted. It isn't always the best strategy to simply try all four directions until the Chariot can get by the monster.

2. THROW SOME GOLD OVERBOARD. When this second option is selected, line 10660 asks the player how much gold he wants to throw. If he tries to throw more gold than he has aboard the Magic Chariot (checked in line 10680), a subroutine from lines 5360 through 5530 is called. The computer selects a random number from 1 to 8 in this subroutine. Each number represents one of the supporting characters aboard the Chariot. The selected character is killed (if he is already dead, nothing happens). The moral here is that you must always keep track of how much gold you are carrying. When facing a monster, you have no way to count your gold pieces.

If the player throws an acceptable number of gold coins overboard, his supply is appropriate decreased, and the goblin greedily eats the gold (line 10940). A random number from 1 to 200 is selected. If this number is less than the number of coins thrown, the goblin dies of terminal indigestion (line 10960) and is erased from both map arrays. If the goblin is not killed, the program returns to line 10320, and the player must select a new action.

3. HAND TO HAND COMBAT. This option was thoroughly discussed earlier. The goblin has a randomly determined strength rating. Either the goblin, or your chosen champion, will be slain. The struggle will weaken the victor somewhat so it is a good idea to vary your champions. Even if your champion is slain, it may be well worthwhile to send out a second champion because the goblin has been weakened.

The outcome of the battle is determined in line 11340, which compares the strength rating of the opponents. The stronger fighter wins.

4. RUB MAGIC ORB. This option is offered in the menu and accepted by the program only if the Magic Orb is aboard the Magic Chariot (MO = 1).

Rubbing the Magic Orb randomly relocates the Chariot somewhere within the Magic Kingdom. Of course, since you could land in an even worse situation (for example, in the midst of the deadly Sirens), this option should be employed only in emergencies.

5. UNSHEATH MAGIC SWORD. The player must possess the Magic Sword (SW = 1) for this action to be offered or accepted. Goblins are instantly slain by the Magic Sword. Other monsters may or may not be killed.

6. PLAY MAGIC ZITHER. Once again, the Magic Zither must be aboard the Chariot (MZ = 1) for this option to be used. Goblins tend not to be very fond of music. If this option is selected when facing a goblin (FO = 3—see line 11640), the subroutine for killing off a character (GOSUB 5360) is called; then a new action choice is requested. Obviously, the Magic Zither is not a very good way to deal with a goblin.

Actually, it is effective for only one type of monster. When the Zither is played for other creatures, the computer simply informs you that "No one is particularly impressed with your talent."

The other monsters are programmed similarly to the goblin. Of course, specific results for each possible action may be different. Each creature must be treated individually. (Incidentally Magic Sparrows are surprisingly strong.)

Table 9.4 Variables Used in the "Golden Flute" Program.

A	misc.
AL	Allegrecia alive?
AM	move allowed by monster
AX	Oracle visit counter
BR	Brombiran alive?

BS, BT, BU	bazooka firing limits
BZ	have Magic Bazooka?
CH	Champion chosen for hand to hand combat
D	character to be killed select
DG	Dagglette alive?
FL	have Golden Flute?
FO	monster currently being faced
G	bazooka firing limits
GL	gold coins
GR	Gromphlur alive?
GW	game counter
H	bazooka firing limits/gold to be tossed overboard/fight potential
I	character's fight power
JS	Jessan alive?
LC	current location
M	move counter
MO	have Magic Orb?
MZ	have Magic Zither?
P	Terak moves?/action against monster
PM	Princess Melva alive?
PW	previous winning score
Q	misc.
QQ	number of collisions with wall
R	revive character select
RU	Rulf alive?
SJ	Sejjan alive?
SW	have Magic Sword?
TT	timing loop counter
U, V, X	misc.
XX	current location value
XZ	display "*" location for explosion
Y, Z	misc.
ZX	crew count
ZZ	misc.

Arrays

A(100)	known map
B(100)	complete map
C(9)	character health ratings

String Variables

N$	player's name
Q$	various inputs

EXPANDING THE GAME

You can expand this game program yourself in a number of ways. One of the most obvious expansions is to add more types of monsters. Use the undefined variable values in the map arrays to place these new creatures you devise. Refer back to Table 9.5 to see which values are already used in the present program.

Real-time actions could be called for when facing a monster. A timing loop and the INKEY$ command could be used as discussed in Chapter 7.

For the GOLDEN FLUTE game you could arrange the real-time programming so that if the player does not make a valid choice of action before the time loop runs out, one of the characters will be killed. The timing loop then starts over. If the player's character is the last one left alive aboard the Magic Chariot when the timing loop runs out, the program could jump to the "YOU ARE DOOMED!" end game message (line 10100).

If you need to conserve memory space, lines 710 through 890 could be eliminated. This section of the program simply prints out the current contents (personnel, weapons, and treasures) of the Magic Chariot after each turn. Since this routine does not directly affect the program in any way, it could be omitted without problems. However, the player must then always keep track of what he's carrying. This frustrates many people. If this section is eliminated, replace line 710 with a REMark statement. If no line exists at 710, the program will bomb.

If you want to speed up the map routine (lines 7000 through 7250), you could use PRINT CHR$(V) statements, rather than the **IF** V = x **THEN PRINT** "XXX" format used in the current program. This would also tend to conserve memory space. You would have to change the values stored in the arrays for the appropriate character codes. The CHR$(V) approach was not used here, because different codes would be required for different computers. Check your manual.

Graphic representations of the various monsters and obstacles could also be included in the program. Refer back to chapter 7 for some hints in this area.

Another possible addition to the game would be to have the player bargain with Terak to get the Golden Flute back. If the player does not have enough coins, Terak keeps the Flute, and moves his lair.

The number of gold coins and/or live characters aboard the Magic Chariot when it returns the Golden Flute to the Woodlands affects the player's final score. This encourages more exploration of the Magic Kingdom (to find more gold) and more conservative strategy (to preserve characters).

Many other additions and plot twists are possible. As with all adventure type games, set your imagination free.

Table 9.5 Map Values Used in the "Golden Flute" Program.

value	meaning	displayed as
0	blank	.
1	the Woodlands	W
2	Coramble, the Great Oracle of Purlicon Mountain	M
3	The Hopeless Pits	P
4	Terak's lair	T
5	Sirens	S
6	dragon	D
7	Magic Sword	.
8	gold coins	.
9	gargoyle	G
10	goblin	g
11	Magic Orb	.
12	The Magic Chariot	C
13, 14	wall	X
15	mysterious fog	F
16	undefined	.
17	Magic Bazooka	.
18, 19	undefined	.
20	smoldering ruin	*
21	Magic sparrow	s
22	ancient dwarf	d
23	Kluffoot	.
24	Friek	.
25	Enchanted Forest	.
26	Magic Zither	.
27	boulder	R
28	enchanted land mine	!
29	wicked witch	w

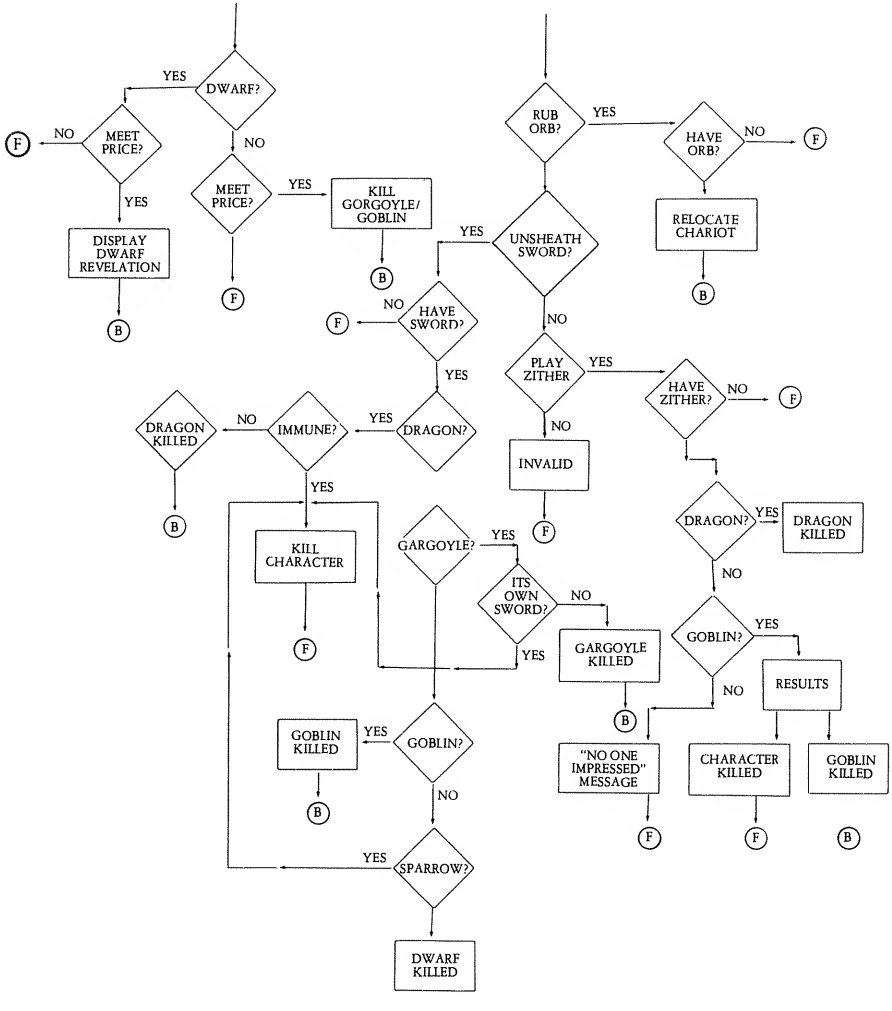


Figure 9.1D Flow Chart for THE GOLDEN FLUTE Program.

Chapter 10

The Great Escape

THE GREAT ESCAPE is a maze game. The player's character is lost in a one-hundred-room building. The object, of course, is to find the way out. This program was written in stages that started with a simple maze game, then added a villain and gold coins. New objects and characters were then added on as they came to mind. Each item was completely programmed before work started on a new item. This prevents the possibility of getting lost and accidentally leaving out essential steps from the program.

Each room within the maze may have up to four doors (labeled "North", "South", "East", and "West"). Some rooms may not have any doors at all. When the player gets stuck, he may use a secret passageway by entering "X" as his move. This will randomly relocate him within the maze. To discourage overuse of this feature, each secret passageway counts as 10 moves.

GOLD COINS AND SCORING

To add interest to the game, gold coins are scattered throughout most of the rooms. Any room may start out with anything from 0 to 10 coins. Weighting is used so that 0 will be the most common single value, yet more than half of the rooms should contain some coins. The number of coins in each room is stored in an array (GC(x)):

```
220 FOR X = 1 TO 100: Y = RND(17)
230 IF Y>10 THEN Y = 0
240 GC(X) = Y: NEXT
```

The player's final score for the game is determined by how many gold coins he is carrying (as many as possible) and how many moves he used to escape from the building (as few as possible). There are one to three exits in the north-most wall (see line 130).

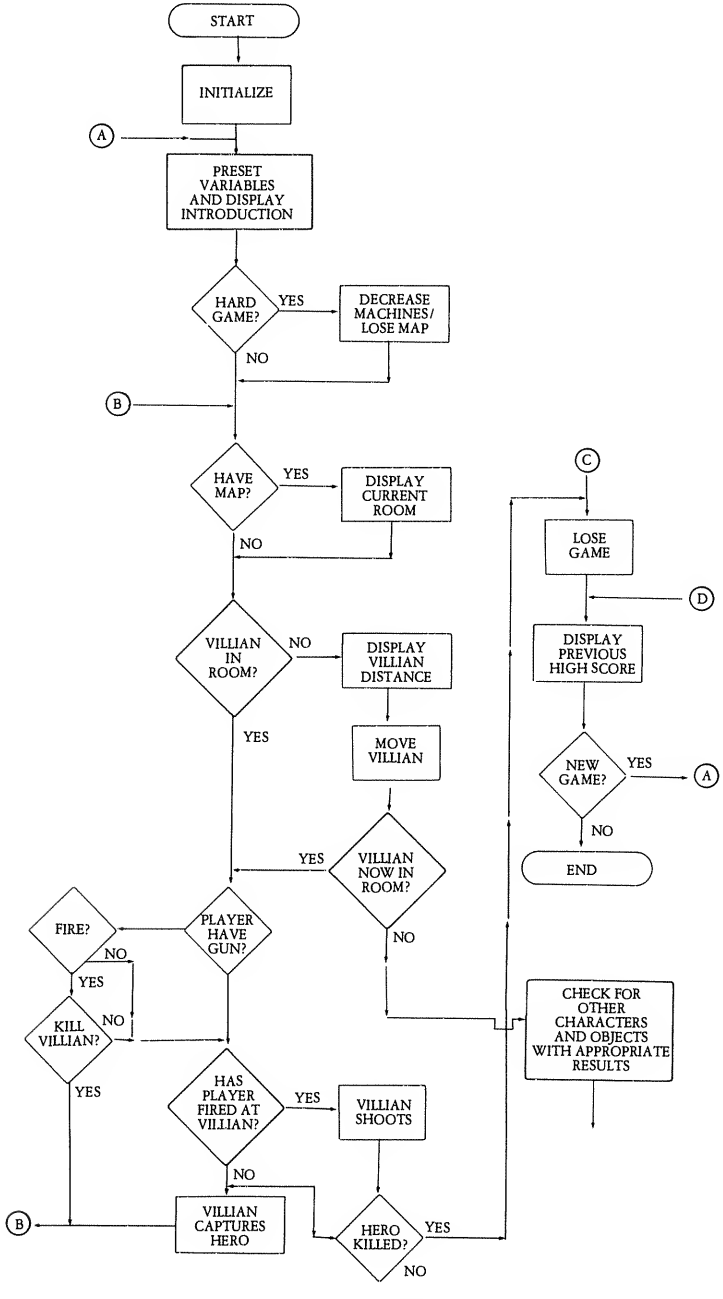


Figure 10.1A Flow Chart for THE GREAT ESCAPE Program.

The scoring formula is calculated in line 5160. It is:

INT(TR/M)*100) – M

where TR represents the number of coins, and M is the number of moves.

For example, let's say the player escaped with 100 coins after 50 moves. The final score would equal INT((100/50) * 100) – 50 = INT(2 * 100) – 50 = 200 – 50 = 150.

Negative scores are also possible. For instance, if the player took 120 moves to get out of the building with 10 coins, his score would work out to INT((10/120)*100) – 120 = INT(0.083333*100) – 120 = 8 – 120 = – 112.

Generally, if the number of coins is less than the number of moves, the score will be negative.

In a series of games, the best previous positive score will be displayed at the end of each game. If the player's character is killed, the final score will not count, even if it is better than the previous high score.

OBSTACLES

As described so far, the game is functional, albeit a bit dull. A player is likely to soon lose interest. The solution, of course, is to add obstacles. If you glance over the complete program listing in Table 10.1, you can see that most of the programming is devoted to the various obstacles and helpers. Just testing for the various additional characters and obstacles takes over 45 lines (from 650 to 1120).

The routines used in the GREAT ESCAPE program are summarized in Table 10.2. A list of the variables and their meanings is included in Table 10.3. A simplified flow chart, shown in figure 10.1, illustrates the basics of the game.

The primary obstacle in the way of the player's escape is a character known as "the villain". The villain tries to capture the escaping hero, and throw him through a trap door to the south-most part of the maze building (the farthest from the exits). The villain will also steal all of the player's gold coins.

On each move, the player is informed of the villain's distance before and afer he moves. For example:

THE VILLAIN IS 17 ROOMS AWAY!
HE MOVES TO FIND YOU!
THE VILLAIN IS 13 ROOMS AWAY!

At random intervals, the computer will also display the villain's exact location (room number). See line 1110.

The player knows how far away the villain is, but (usually) not which direction. The villain, on the other hand, knows whether the hero is in

front of him or behind him, but not the distance. Each time the villain moves, he jumps from 1 to 10 rooms in the correct direction.

Table 10.1 Complete "The Great Escape" Program.

```
 10  CLEAR 100
 20  PRINTCHR$(21)
 30  REM * THE GREAT ESCAPE * Delton T. Horn *
 40 DIM T(10): DIM N(100): DIM S(100): DIM E(100): DIM W(100)
 50 DIM BT(12): DIM CS(54): DIM CR(52): DIM GC(100): G4 = 1
 69  REM * SHUFFLE CARDS  :
 70  FOR X = 1 TO 52: CS(X) = 0: CR(X) = 0: NEXT
 80  FOR X = 1 TO 4: FOR Y = 1 TO 13
 90  PRINT"*";: Z = RND(52): IF CS(Z) > 0 GOTO 90
100  CS(Z) = X: CR(Z) = Y + 1: NEXT: PRINT: NEXT
110  CLS: PRINT: PRINT: PRINT"  ","THE GREAT ESCAPE":
     PRINT"  ","  ",by Delton T. Horn"
119  REM * SET BUILDING *
120  FOR X = 1 TO 100: N(X) = 0: S(X) = 0: E(X) = 0: W(X) = 0: NEXT
130  FOR X = 1 TO 3: Y = RND(10): N(Y) = 1: NEXT
140  FOR X = 1 TO 65: Y = RND(90): Z = Y + 10: N(Z) = 1: S(Y) = 1: NEXT
150  PRINT: PRINT"You are lost in a giant maze of 100 rooms",CHR$(197)
160  FOR X = 1 TO 73: Y = RND(99): Z = Y + 1: E(Z) = 1: W(Y) = 1: NEXT
170  PRINT"The exit is to the north."
180  FOR X = 1 TO 10: YY = RND(100): BT(X) = YY
190  Y = X*10: Z = Y - 9: E(Z) = 0: W(Y) = 0: NEXT
200  PRINT: PRINT"Find your way out -  ";
210  FOR X = 1 TO 10: T(X) = RND(100): NEXT
220  FOR X = 1 TO 100: Y = RND(17)
230  IF Y > 10 THEN Y = 0
240  GC(X) = Y: NEXT
250  PRINT"  if you can!!!": PRINT
260  T5 = RND(100): K1 = RND(60) + 30: IF T5 = K1 GOTO 260
270  HA = 0: W1 = RND(100): W2 = RND(100): IF W1 = W2 GOTO 270
280  G4 = G4 + 1: MF = 100: SC = RND(100): RB = RND(100): IF SC = RB
     GOTO 280
290  DF = 1: MC = RND(40) + 10: MG = RND(50) + 10: GF = RND(50) + 40
300  IF(MC = MG) OR (MC = GF) OR (RB = MC) OR (RB = MG) GOTO 290
310  BX = 0: BB = RND(100): GR = RND(100): BN = RND(100): IF GR = BN
     310
320  PRINT"If you get stuck you can use a secret passageway by entering"
330  CC = 1: H = 0: BT = 150: BR = RND(70) + 30
340  P = RND(50) + 40: M = 1: V2 = 0: GH = RND(20) + 10
350  LM = RND(100): IF LM = GH GOTO 350
```

```
360 PRINT"'X' as your move."
370 LQ = RND(40)+60: QW = ABS(LQ - P): IF QW < 15 GOTO 370
380 ML = RND(90)+10: L2 = RND(80)+6: IF L2 = ML GOTO 380
390 V = RND(100): IF V = P GOTO 390
400 PRINT"A secret passageway will count as 10 moves."
410 N = 0: GN = RND(90)+10.6: LH = 0: LQ = RND(45)+55:
    SW = RND(100)
420 PD = RND(100): K = RND(40)+60: S = RND(90)+10
430 PRINT: INPUT"Would you like to try a hard game";Q$
440 Q$ = LEFT$(Q$,1)
450 IF Q$ = "Y" GOSUB 10000
499 REM * THE MAIN PLAY *
500 IF P > 100 THEN P = RND(10)+90
510 CLS: PRINT: PRINT: PRINT"MOVE #";M,
520 IF LH = 0 PRINT"You are now in room #";P ELSE PRINT
530 O = 0: FOR U = 1 TO 10: IF BT(U) = P THEN O = 1
535 NEXT
540 IF O = 1 GOSUB 10030
550 IF V > 100 THEN V = RND(20)+80
560 IF TR = 0 GOTO 580
570 PRINT"You are carrying  ";TR;"  gold coin";: IF TR > 1 PRINT"s" ELSE
    PRINT
580 IF V < 1 GOTO 650
590 GOSUB 10180: IF V = P GOTO 5000
600 PRINT"He moves to find you!": W = RND(10)
610 IF V > P THEN V = V - W ELSE V = V + W
620 GOSUB 10180
630 IF V = P GOTO 5000
649 REM * SPECIAL TESTS *
650 IF BX = 0 GOTO 680
660 PRINT"You are carrying a mysteriously ticking box."
670 IF (BX < M) OR (BX = M) GOTO 5220
680 IF SC = 0 PRINT"You are carrying a screwdriver."
690 IF SW = 0 PRINT"You are carrying a saw."
700 IF BN = P GOSUB 10530
710 IF BN = 0 GOSUB 10570
730 IF K1 = 0 PRINT"You are carrying a key."
740 IF BB = P GOSUB 10270
750 IF SC = P PRINT"You just found a screwdriver.": SC = 0
760 IF K1 = P PRINT"You just found a key  ": K1 = 0
770 IF GC(P) = 0 GOTO 810
780 PRINT"You just found  "; GC(P);"  gold coin";
790 IF GC(P) > 1 PRINT"s";
800 PRINT"!": TR = TR+GC(P): GC(P) = 0
810 IF SW = P GOSUB 10310
```

```
820 IF GN<1 GOSUB 10390
830 IF INT(GN) = P GOSUB 10350
840 IF K = P GOSUB 10420
850 IF TR = P PRINT"Your gold coins are magically doubled!": TR = 2*TR
860 IF LM = P GOTO 5250
870 IF L2 = P GOTO 5450
880 X = RND(45): IF ((X>40) AND (LH = 0)) OR ((X>30) AND (LH = 1))
    GOSUB 10500
890 IF (N>0) AND ((N = M) OR (N<M)) PRINT"It's too late!": GOTO 5130
900 IF N>0 PRINT"You must find the first aid kit within  ";N - M;"  moves."
910 IF P = PD GOTO 5620
920 IF P = -PD PRINT"There is a dead puppy dog in this room."
930 IF BT = P GOSUB 10620
940 IF DF = P GOSUB 10650
950 IF MF = P GOSUB 10670
960 DF = DF+1: MF = MF-.5
970 IF P = GF GOSUB 10700
980 IF P = S GOSUB 5800
990 IF P = -RB PRINT"There is a pile of useless metallic junk on the floor."
1000 IF P = RB GOTO 5900
1010 IF P = MG GOTO 6450
1020 IF P = MC GOTO 6680
1030 IF P = GH GOTO 7000
1040 IF P = GR GOTO 7150
1050 IF P = W1 GOTO 7430
1060 IF P = W2 GOTO 7460
1070 IF P = T5 GOTO 7600
1075 IF P = -T5 PRINT"There is an empty treasure chest here."
1080 IF LQ = 0 GOTO 1090
1090 IF LQ = P GOSUB 10920 ELSE GOSUB 10960
1100 IF (BT>0) AND (BT<25) PRINT"Your flashlight is getting dim."
1110 VS = RND(7): IF (VS>5) AND (V>0) PRINT"The villain is now lurking in
    room #";V
1120 GOSUB 10750
1130 PRINT"  ","*** AVAILABLE EXITS ***"
1140 IF BT<1 PRINT"????",: GOTO 1190
1150 IF N(P) = 1 PRINT"North",
1160 IF S(P) = 1 PRINT"South",
1170 IF E(P) = 1 PRINT"East",
1180 IF W(P) = 1 PRINT"West",
1190 IF SW = 0 PRINT"Cut new exit",
1200 PRINT: PRINT"  ",: INPUT"YOUR MOVE";M$: M$ = LEFT$(M$,1)
1210 M = M+1: BT = BT-1
1220 IF M$ = "X" GOTO 4000
1230 IF (M$ = "N") AND (N(P) = 1) GOTO 4020
```

```
1240 IF (M$ = "S") AND (S(P) = 1) GOTO 4030
1250 IF (M$ = "E") AND (E(P) = 1) GOTO 4040
1260 IF (M$ = "W") AND (W)P) = 1) GOTO 4050
1270 IF (M$ = "C") AND (SW = 0) GOTO 4060
1280 HA = HA + 1: PRINT"YOU JUST RAN INTO A WALL, KLUTZ!"
1290 IF HA < 50 GOTO 520
1300 PRINT"You have walked into so many walls, you suffer brain damage!"
1310 IF N > 0 GOTO 5130
1320 N = M + 75: PRINT"You must find the first aid kit within 75 moves or you"
1330 PRINT"will lapse into a permanent coma.": GOTO 520
4000 PRINT: PRINT"  ","SECRET PASSAGEWAY": PRINT: PRINT
4010 BT = BT - 9: FOR X = 1 TO 222: NEXT X: P = RND(100): M = M + 9:
     GOTO 500
4020 P = P - 10: IF P < 1 GOTO 4200 ELSE GOTO 500
4030 P = P + 10: GOTO 500
4040 P = P - 1: IF P < 1 GOTO 4200 ELSE GOTO 500
4050 P = P + 1: GOTO 500
4060 INPUT"Which wall do you cut a hole in";Q$: Q$ = LEFT$(Q$,1)
4070 IF (Q$ = "N") OR (Q$ = "S") OR (Q$ = "E") OR (Q$ = "W") GOTO 4090
4080 PRINT"THERE IS NO SUCH WALL.": GOTO 1130
4090 X = RND(20): IF X > 15 GOTO 4150
4100 IF Q$ = "N" THEN N(P) = 1
4110 IF Q$ = "S" THEN S(P) = 1
4120 IF Q$ = "E" THEN E(P) = 1
4130 IF Q$ = "W" THEN W(P) = 1
4140 IF X > 9 GOTO 4170 ELSE GOTO 1130
4150 PRINT"The saw breaks before you make an opening large enough to"
4160 PRINT"crawl through.": SW = - 1: GOTO 1130
4170 PRINT"The saw breaks just as you finish cutting the new exit."
4180 SW = - 1: GOTO 1130
4200 PRINT"  ","YOU MADE IT!!!"
4210 PRINT"It took you  ";M;"  moves, and you got out with  ";TR;"  coins!"
4220 GOTO 5160
4998 STOP
4999 REM * VILLAIN ENCOUNTER *
5000 IF GN < 1 GOTO 5050
5010 IF H = 1 GOTO 5080
5020 PRINT"He captures you and throws you through a secret passageway!"
5030 IF TR > 0 PRINT"He also steals all of your gold coins!": V2 = V2 + TR:
     TR = 0
5040 P = RND(10) + 90: GOTO 650
5050 INPUT"Do you shoot at him";Q$: Q$ = LEFT$(Q$,1)
5060 X = 0: IF Q$ = "Y" GOTO 5070
5065 GOTO 5010
5070 GOSUB 10210: H = 1: IF X > 6 GOTO 5210 ELSE PRINT"You missed!":
     GOTO 5010
```

```
5080 PRINT"He pulls out his own gun and shoots at you!": FOR X = 1 TO 222:
     NEXT
5090 PRINT"  ".""*** BANG! ***": FOR X = 1 TO 222: NEXT: PRINT:
     X = RND(10)
5100 IF X > 7 GOTO 5120
5110 PRINT"WHEW! He missed!": GOTO 5020
5120 PRINT"HE GOT YOU!": PRINT
5130 PRINT"You are deceased. . .": PRINT
5140 PRINT: PRINT"You survived  ";M;"  moves, and had  ";TR;"  gold
     coins"
5150 RT = 0: GOTO 5170
5160 RT = INT((TR/M)*100) – M: PRINT"Your score this time was  ";RT
5170 IF RX = 0 THEN RX = RT: GOTO 5200
5180 PRINT"Previous high score was  ";RX
5190 IF RT > RX THEN RX = RT
5200 PRINT"Play game #";G4;: INPUT Q$: Q$ = LEFT$(Q$,1): IF Q$ = "Y"
     GOTO 110
5205 PRINTCHR$(21): END
5210 PRINT"GOT 'EM!!": V = 0: GOTO 650
5220 PRINT"The time bomb in the mysteriously ticking box explodes!"
5230 FOR X = 1 TO 222: NEXT: PRINT"  ","Ka – ";
5240 FOR X = 1 TO 234: NEXT: PRINT"BOOM!!": PRINT: GOTO 5130
5249 REM * MAGIC LAMP 1 *
5250 GOSUB 10450
5260 IF Q$ = "Y" GOTO 5280
5270 GOTO 870
5280 LM = LM + RND(5): ML = RND(9)
5300 IF(ML = 1) AND (GN < 1) GOTO 5280
5310 IF ML = 1 PRINT"You are transportred to the room with the gun.":
     P = INT(GN)
5320 IF ML = 2 PRINT"You are transported to the room with the first aid kit.":
     P = K
5330 IF (ML = 3) AND (SW < 1) GOTO 5280
5340 IF ML = 3 PRINT"You are transported to the room with the saw.": P = SW
5350 IF ML = 4 PRINT"The villain is placed in room #100.": V = 100
5360 IF ML = 5 THEN P = RND(20): PRINT"You are transported to room #";P
5370 IF (ML = 6) AND (GH = 0) GOTO 5280
5380 IF ML = 6 PRINT"The ghost has been exorcised.": GH = 0
5390 IF ML = 7 PRINT"Your move count has been reduced.":
     M = INT(M – RND(M/4))
5400 IF (ML = 8) AND (LH = 0) GOTO 5280
5410 IF ML = 8 PRINT"A map appears.": LH = 0
5420 IF ML = 9 PRINT"A set of spare batteries appears.": BT = BT + 100
5430 PRINT: INPUT"Please press 'ENTER'  ";Q$
5440 GOTO 500
```

```
5450  GOSUB 10450
5460  IF Q$ = "Y" GOTO 5480
5470  GOTO 880
5480  L2 = 0: ML = RND(12)
5490  IF ML = 1 PRINT"Dame Fortune is dead.": DF = 101
5500  IF ML = 2 PRINT"Miss Fortune is dead.": MF = 0
5510  IF ML = 3 PRINT"The leprechaun is dead.": LQ = 0
5520  IF ML = 4 PRINT"The villain is dead.": V = 0
5530  IF ML = 5 PRINT"The robot's batteries are dead.": RB = 0
5540  IF ML = 6 PRINT"The MAD GAMBLER is dead.": MG = 0
5550  IF ML = 7 PRINT"The ghost has been returned to its grave.": GH = 0
5560  IF ML = 8 PRINT"The evil merchant is dead.": MC = 0
5570  IF ML = 9 PRINT"The puppy dog is dead.": PD = -PD
5580  IF ML = 10 PRINT"The gorilla is dead.": GR = 0
5590  IF ML = 11 PRINT"The good fairy is dead.": GF = 0
5600  IF ML = 12 GOTO 5130
5610  GOTO 880
5620  PRINT"There is a puppy dog in this room.": X = RND(10): Y = 20
5630  IF X > 7 PRINT"  ","WOOF! WOOF!"
5640  IF GN > .99 GOTO 5710
5650  INPUT"Do you shoot the puppy dog";Q$: Q$ = LEFT$(Q$,1)
5660  IF Q$ = "Y" GOTO 5680
5670  GOTO 5710
5680  PRINT"Oh, you are a mean person.": GOSUB 10210
5690  IF (X > 0) AND (X < 7) THEN PD = -PD: GOTO 910
5700  PRINT"You missed.": Y = 10
5710  X = RND(Y): IF X < 4 GOTO 5740
5720  IF X > 7 PRINT"The puppy dog wags its cute li'l tail."
5730  GOTO 920
5740  PRINT"The puppy dog bites you!": IF N > 0 GOTO 5760
5750  N = 100 + M: GOTO 5780
5760  X = N - M: IF X < 4 GOTO 5130
5770  N = M + INT(X/2)
5780  PRINT"You must find the first aid kit within  ";N - M;"  moves."
5790  GOTO 920
5800  PRINT"There is a snake in this room.": PRINT"  ","hssss. . .": PRINT
5810  IF GN < 1 GOTO 5860
5820  PRINT"You are bitten!": IF N > 0 GOTO 5130
5830  PRINT"There is some antitoxen in the first aid kit."
5840  PRINT"You must find it within 50 moves or die.": N = M + 30
5850  X = RND(10): IF X > 5 PRINT"The snake slithers off.": S = RND(100)
5855  GOTO 990
5860  INPUT"Do you try to shoot the snake";Q$: Q$ = LEFT$(Q$,1)
5870  IF Q$ = "Y" GOSUB 10210 ELSE GOTO 5820
5880  IF X > 5 PRINT"You missed!": GOTO 5820
```

```
5890 PRINT"GOT 'EM!": S = 0: GOTO 990
5900 PRINT"There is an eight foot tall robot in this room!"
5910 PRINT"  ","* beep *","  ","* beep *": PRINT
5920 RG = RND(50)
5930 PRINT"  ","What do you do?": PRINT"X — Use secret passageway"
5940 IF BN = 0 PRINT"F — Feed bananas to robot"
5950 IF GN < 1 PRINT"S — Shoot robot"
5960 PRINT"P — Push past robot to the nearest exit"
5970 IF SC = 0 PRINT"D — Dismantle robot"
5980 IF TR > 0 PRINT"G — Give some gold coins to the robot"
5990 PRINT"C — Cry": PRINT"E — Eat robot"
6000 INPUT"Your selection";Q$: Q$ = LEFT$(Q$,1)
6010 IF Q$ = "X" GOTO 4000
6020 IF (Q$ = "F") AND (BN = 0) GOTO 6100
6030 IF (Q$ = "S") AND (GN < 1) GOTO 6130
6040 IF Q$ = "P" GOTO 6170
6050 IF (SC = 0) AND (Q$ = "D") GOTO 6290
6060 IF (TR > 0) AND (Q$ = "G") GOTO 6320
6070 IF Q$ = "C" GOTO 6400
6080 IF Q$ = "E" GOTO 6430
6090 PRINT"That is not an acceptable action.": GOTO 5930
6100 PRINT"The robot takes the bananas in its gripper"
6110 FOR X = 1 TO 232: NEXT: PRINT"And squeezes them into a disgusting
     pulp."
6120 FOR X = 1 TO 246: NEXT: PRINT: PRINT"Robots do not eat bananas.":
     GOTO 5930
6130 GOSUB 10210: IF X = 0 GOTO 5930
6140 PRINT"The bullet bounces off the robot's metallic hide without even"
6150 PRINT"denting it.": IF X < 9 GOTO 5930
6160 PRINT"But it hits you on the richochet!": GOTO 5130
6170 PRINT"The robot will not let you push by.": PRINT"It breaks your  ";
6180 X = RND(7): IF X = 1 PRINT"arm": Y = 100: XY = 5
6190 IF X = 2 PRINT"leg": Y = 75: XY = 8
6200 IF X = 3 PRINT"nose": Y = 80: XY = 6
6210 IF X = 4 PRINT"thumb": Y = 150: XY = 2
6220 IF X = 5 PRINT"back": Y = 35: XY = 17
6230 IF X = 6 PRINT"neck": Y = 30: XY = 20
6240 IF X = 7 PRINT"big toe": Y = 150: XY = 3
6250 IF N > 0 GOTO 6280
6260 N = Y + M: PRINT"You must find the first aid kit
     within  ";N - M;"  moves"
6270 GOTO 5930
6280 N = N - XY: IF N < M GOTO 5130 ELSE GOTO 5930
6290 PRINT"The screwdriver comes in handy. You soon reduce the robot to"
6300 PRINT" a heap of useless junk.": RB = - RB
```

```
6310 GOTO 1010
6320 INPUT"How many coins do you offer to the robot";X
6330 IF X>TR GOTO 6370
6340 TR=TR-X: PRINT"The robot takes ";X;" gold coins and ";
6350 IF X>RG PRINT"leaves": RB=0: GOTO 1010
6360 PRINT"blinks a few lights at you.": GOTO 5930
6370 PRINT"You do not have ";X;" coins!"
6380 TR=0: PRINT"The robot takes the coins you do have."
6390 PRINT"Then it breaks your ";: GOTO 6180
6400 PRINT"Crying will do you no good."
6410 PRINT"Did you think the robot would take pity on you?"
6420 PRINT"Ha! It has a heart of tin.": GOTO 5930
6430 PRINT"You can't eat a robot, you silly person.": X=RND(12)+1
6440 PRINT"You only succeed in breaking ";X;" teeth.": GOTO 5930
6450 PRINT"You just met up with the MAD GAMBLER!": PRINT
6460 INPUT"Please press 'ENTER' ";Q$: CLS: IF TR>10 GOTO 6490
6470 PRINT"Because you have so few coins, he has no interest"
6480 PRINT"in you. He throws you through a trap door.": P=RND(20)+80
6485 INPUT"Please press 'ENTER' ";Q$: GOTO 510
6490 PRINT"MAD GAMBLER: He-he-hee! Greetings, my pigeon!"
6500 PRINT" ","We shall play A little High Card."
6510 PRINT" ","If you draw the high card, I'll let you"
6520 PRINT" ","exit to the north, and I'll double your"
6530 PRINT" ","gold coins. But if I win, I get half"
6540 PRINT" ","your coins, and we play again."
6550 FOR X=1 TO 234: NEXT: PRINT: PRINT"You draw—";
6560 GOSUB 10820
6570 X1=CS(CC): X2=CR(CC): CC=CC+1
6580 GOSUB 10820
6590 Y1=CS(CC): Y2=CR(CC): CC=CC+1
6600 IF X1>Y1 GOTO 6650
6610 IF (X1=Y1) AND (X2>Y2) GOTO 6650
6620 PRINT" ","YOU LOSE!!!": PRINT"MAD GAMBLER: Ha! Ha! Ha!":
     TR=INT(TR/2): IF TR<8 GOTO 6470
6630 PRINT" ","Let's play again, pigeon."
6640 INPUT"Press 'ENTER' to draw ";Q$: GOTO 6550
6650 PRINT" ","YOU WIN!!": PRINT"MAD GAMBLER: Rats!"
6660 P=P-10: TR=TR*2: INPUT"Press 'ENTER' to exit to the North ";Q$
6670 GOTO 500
6680 PRINT"You encounter an evil merchant.": IF TR<10 GOTO 6470
6690 PRINT"EVIL MERCHANT: Well, my friend—you may exit to the"
6700 PRINT" ","south, or you can bargain with me for"
6710 PRINT" ","an exit to the north. Do you want to"
6720 PRINT" ","bargain";: INPUT Q$: Q$=LEFT$(Q$,1)
6730 IF Q$="Y" GOTO 6770
```

```
6740 PRINT: PRINT"EVIL MERCHANT: I'm sorry we couldn't do business
     together."
6750 PRINT: INPUT"Press 'ENTER' to exit to the south";Q$
6760 P = P + 10: GOTO 500
6770 X = RND(TR): CO = TR
6780 PRINT: PRINT"EVIL MERCHANT: Make me an offer, my friend."
6790 PRINT"  ","(enter 0 to end bagaining)"
6800 INPUT"YOUR OFFER";Y: IF Y = 0 GOTO 6740
6805 IF Y > TR GOTO 6840
6810 IF Y < X GOTO 6880
6820 PRINT"EVIL MERCHANT: All right. I'll accept that, my friend."
6830 P = P - 10: INPUT"Press 'ENTER' to exit to the north  ";Q$: TR = TR - Y:
     GOTO 500
6840 PRINT"EVIL MERCHANT: Bah! You do not have  ";Y;"  gold coins!"
6850 PRINT"  ","I do not enjoy being cheated!": PRINT
6860 PRINT"He takes what coins you do have and throws you through a trap"
6870 TR = 0: P = RND(20) + 80: INPUT"door. Press 'ENTER'  ";Q$: GOTO 500
6880 PRINT"EVIL MERCHANT:  ";: XY = RND(7)
6890 IF XY = 1 PRINT"Bah! An exit to the north is worth at least  ";CO
6900 IF XY = 2 PRINT"No, do I look like the kind of man who":
     PRINT"  ","would accept  ";Y;"?"
6910 IF XY = 3 PRINT"Bah! Chicken feed."
6920 IF XY = 4 PRINT"No. I want  ";CO;"  at the very least."
6930 IF XY = 5 PRINTCO;"  might be reasonable, but not  ";Y
6940 IF XY = 6 PRINT"No. Make me a reasonable offer."
6950 IF XY = 7 PRINT"Never would I sell an exit to the north": PRINT"  ","for
     a mere  ";Y
6960 CO = CO - RND(10): IF CO < X THEN CO = X
6970 GOTO 6780
7000 PRINT: PRINT"  ","BOO!": PRINT
7010 PRINT"This room is haunted by a ghost!"
7020 IF GN < 1 GOTO 7080
7030 INPUT"YOUR MOVE";M$: M$ = LEFT$(M$,1)
7040 M = M + 1: IF M$ = "X" GOTO 4000
7050 IF (M$ = "N") OR (M$ = "S") OR (M$ = "E") OR (M$ = "W") GOTO 7070
7060 PRINT"WHAT???": GOTO 7030
7070 PRINT"The ghost will not let you go that way.": GOTO 7030
7080 INPUT  "Do you shoot at the ghost";Q$: Q$ = LEFT$(Q$,1)
7090 IF Q$ = "Y" GOTO 7110
7100 GOTO 7030
7110 GOSUB 10210
7120 IF X = 0 GOTO 7030
7130 PRINT"The bullet passes harmlessly through the ghost."
7140 PRINT"It's still there.": GOTO 7080
7150 PRINT"A gorilla is in this room!"
```

```
7160  IF GN<1 GOTO 7300
7170  IF BN=0 GOTO 7380
7180  X=M+3
7190  INPUT"Your move";M$: M$=LEFT$(M$,1)
7200  M=M+1: IF M$="X" GOTO 4000
7210  IF (M$="N") OR (M$="S") OR (M$="W") OR (M$="E") GOTO 7240
7220  PRINT"WHAT?": IF M=X GOTO 7260
7230  GOTO 7190
7240  PRINT"The gorilla will not let you go that way."
7250  IF M<X GOTO 7190
7260  PRINT"Annoyed by your fooling around, the gorilla whups you"
7270  PRINT"upside the head -- and this monkey is STRONG!"
7280  IF N>0 GOTO 5130
7290  N=M+50: PRINT"You must find the first aid kit within 50 moves!":
      GOTO 7190
7300  INPUT"Do you shoot at the gorilla";Q$: Q$=LEFT$(Q$,1)
7310  IF Q$="Y" GOTO 7330
7320  GOTO 7170
7330  GOSUB 10210
7340  IF X>5 GOTO 7260
7350  PRINT"HA! You got the big ape!"
7360  GR=0: GOTO 1050
7380  INPUT"Do you try feeding the bananas to the gorilla";Q$
7390  Q$=LEFT$(Q$,1): IF Q$="Y" GOTO 7410
7400  GOTO 7180
7410  PRINT"The gorilla leaves, happily munching on the bananas."
7420  BN=-1: GR=0: GOTO 1050
7430  PRINT"A wicked witch rearranges some of the trap doors ": PRINT"in the
      building!"
7440  Y=RND(9)+1: FOR X=1 TO Y: T(X)=RND(100): NEXT
7450  GOTO 1060
7460  PRINT"A wicked witch rearranges some of the doors in"
7470  PRINT"the building!"
7480  FOR X=1 TO 75: Y=RND(80)+10: XY=RND(5)
7490  IF XY=1 THEN N(Y)=0
7500  IF XY=2 THEN S(Y)=0
7510  IF XY=3 THEN E(Y)=0
7520  IF XY=4 THEN W(Y)=0
7530  Y=RND(80)+10: XY=RND(4)
7540  IF XY=1 THEN N(Y)=1
7550  IF XY=2 THEN S(Y)=1
7560  IF XY=3 THEN E(Y)=1
7570  IF XY=4 THEN W(Y)=1
7580  NEXT: GOTO 1070
```

```
7600 PRINT"There is a treasure chest in this room."
7610 PRINT"But it is locked.": IF GN<1 GOTO 7650
7620 IF K1=0 GOTO 7730
7630 GOTO 1080
7650 PRINT"Do you try to shoot the lock off";Q$: Q$=LEFT$(Q$,1)
7660 IF Q$="Y" GOTO 7680
7670 GOTO 7620
7680 GOSUB 10210
7690 IF X<7 GOTO 7620
7700 PRINT"THE TREASURE CHEST IS OPEN!!": TX=RND(888)
7710 PRINT"It contains ";TX;" gold coins!"
7720 T5=-T5: TR=TR+TX: TX=0: GOTO 1080
7730 INPUT"Do you try using the key";Q$: Q$=LEFT$(Q$,1)
7740 IF Q$="Y" GOTO 7700 ELSE GOTO 1080
9999 STOP
10000 LH=1: FOR X=1 TO 5: BT(X)=0: NEXT
10010 L2=0: DF=50: GF=0
10020 FOR X=1 TO 45: YX=RND(80)+10: N(YX)=0: NEXT: RETURN
10030 IF M<10 THEN RETURN
10040 PRINT: PRINT"There is a vending machine in this room."
10050 PRINT" ","Insert 25 coins for fresh batteries!"
10060 INPUT"Do you use the machine";Q$: Q$=LEFT$(Q$,1)
10070 IF Q$="Y" GOTO 10090
10080 PRINT"OK": RETURN
10090 IF TR<25 GOTO 10120
10100 TR=TR-25: BT=125: PRINT"You now have a fresh set of batteries."
10110 RETURN
10120 IF TR>10 GOTO 10150
10130 PRINT "For trying to cheat the machine, you are sent to room #100!"
10140 TR=0: P=100: RETURN
10150 TR=0: PRINT"The machine eats all your coins, but doesn't give you
      anything"
10160 PRINT"because you cannot meet the price."
10170 RETURN
10180 W=ABS(V-P): PRINT"Villain is ";W;" room";
10190 IF (W=0) OR (W>1) PRINT"s";
10200 PRINT" away from you.": RETURN
10209 REM * SHOOT *
10210 IF GN<0.1 GOTO 10250
10220 PRINT: PRINT"*** BANG! ***": PRINT
10230 X=RND(10): GN=GN-0.1
10240 RETURN
10250 PRINT: PRINT"*** click ***": PRINT
10260 X=0: RETURN
10270 PRINT"There is a mysteriously ticking box in this room.": PRINT
```

```
10280  INPUT"Do you pick it up";Q$:Q$ = LEFT$(Q$,1)
10290  IF Q$ = "Y" THEN BB = 0: BX = M + 100
10300  RETURN
10310  PRINT"There is a saw lying in the corner.": PRINT
10320  INPUT"Do you pick it up";Q$: Q$ = LEFT$(Q$,1)
10330  IF Q$ = "Y" THEN SW = 0
10340  RETURN
10350  PRINT"There is a gun on the floor here.": PRINT
10360  INPUT"Do you pick it up";Q$: Q$ = LEFT$(Q$,1)
10370  IF Q$ = "Y" THEN GN = GN - INT(GN)
10380  RETURN
10390  PRINT"You are carrying a gun with  ";INT(GN*10);
10400  PRINT"  bullet";: IF GN = .1 PRINT"." ELSE PRINT"s."
10410  RETURN
10420  PRINT"There is a first aid kit in this room."
10430  IF N > 0 PRINT"  ","YOU ARE SAVED!!"
10440  N = 0: RETURN
10450  PRINT"There is a magic lamp chained to the wall.": PRINT
10460  INPUT"Do you rub the lamp";Q$: Q$ = LEFT$(Q$,1)
10470  IF Q$ = "Y" GOTO 10490
10480  RETURN
10490  CLS: PRINT: PRINT: PRINT"  ","* POOF! *": PRINT: PRINT"The lamp
       vanishes.": RETURN
10500  IF LH = 1 GOTO 10520
10510  PRINT"Oh oh. You just lost your map.": LH = 1: RETURN
10520  PRINT"Hey! You just found a map.": LH = 0: RETURN
10530  PRINT"You just found a bunch of ripe bananas."
10540  INPUT"Do you pick them up";Q$: Q$ = LEFT$(Q$,1)
10550  IF Q$ = "Y" THEN BN = 0
10560  RETURN
10570  PRINT"You are carrying a bunch of bananas. Do you want to eat them";
10580  INPUT Q$:Q$ = LEFT$(Q$,1): IF Q$ = "Y" GOTO 10600
10590  RETURN
10600  BN = - 1: PRINT: PRINT"  ","burp": PRINT: IF N = 0 THEN RETURN
10610  N = N + 50: RETURN
10620  PRINT"There is a battery recharger in this room."
10630  IF BT < 10 THEN BT = 100 ELSE BT = BT + 50
10640  RETURN
10650  PRINT"You just ran into Dame Fortune!": TR = TR + 100
10660  P = RND(20): RETURN
10670  PRINT"You just ran into Miss Fortune!": P = 100
10680  IF TR > 0 THEN TR = INT(TR/2)
10690  RETURN
10700  PRINT"A good fairy plants more gold coins throughout the building!"
10710  PRINT: GF = 0: FOR X = 1 TO 100: Y = RND(12): IF Y > 10 THEN Y = 0
```

```
10720  GC(X) = GC(X) + Y: NEXT: GC(P) = 0: RETURN
10730  PRINT"The batteries in your flashlight are dead."
10740  PRINT"You cannot see where you are going.": RETURN
10750  FOR X = 1 TO 10: IF T(X) = P GOTO 10770
10760  NEXT: RETURN
10770  PRINT"You just fell through a trap door!"
10780  IF P > 80 GOTO 10810
10790  Y = RND(50): XY = P + Y: IF XY > 100 GOTO 10790
10800  P = P + Y: GOTO 10760
10810  P = RND(20) + 80: GOTO 10760
10820  IF CR(CC) < 11 PRINTCH(CC);
10830  IF CR(CC) = 11 PRINT"JACK";
10840  IF CR(CC) = 12 PRINT"QUEEN";
10850  IF CR(CC) = 13 PRINT"KING";
10860  IF CR(CC) = 14 PRINT"ACE";
10870  PRINT"  OF  "; IF CS(CC) = 1 PRINTCHR$(192)
10880  IF CS(CC) = 2 PRINTCHR$(194)
10890  IF CS(CC) = 3 PRINTCHR$(195)
10900  IF CS(CC) = 4 PRINTCHR$(193)
10910  RETURN
10920  PRINT"You just caught the leprechaun!"
10930  IF TR < 10 THEN TR = 50 ELSE TR = INT(TR*2.5)
10940  LQ = RND(60) + 30: X = ABS(LQ - P): IF X < 20 GOTO 10940
10950  RETURN
10960  X = ABS(LQ - P): PRINT"The leprechaun is  ";X;"  room";
10970  IF X > 1 PRINT"s away": ELSE PRINT"  away"
10980  PRINT"But then he moves"
10990  LQ = LQ + RND(5): IF LQ > 100 THEN LQ = RND(60) + 30
11000  X = ABS(LQ - P): IF LQ < 15 GOTO 10990
11010  RETURN
```

Of course, he will often over-shoot and go past the player's position. For example, if the villain is three rooms in front of the hero, he may move back five and end up two rooms behind the hero.

In spite of this, the villain will catch the hero more often than you might expect. When the villain and the player are in the same room the hero is immediately captured, unless he is carrying the gun (which is hidden in one of the rooms at the beginning of the game).

If he has the gun, the player is asked if he wants to shoot the villain (lines 5050 through 5065). The shooting is done in a subroutine that begins at line 10210. Notice that this subroutine checks to make sure the gun has bullets. A random number (X) from 1 to 10 is selected (line 10230) to represent the results of the shot. If this number is greater than 6 (line 5070), the villain is killed (line 5210). For values of X that are six or less, the

player's shot misses, and the villain pulls out his own gun and shoots back (lines 5080 through 5120).

The hero has a 40 percent chance of killing the villain, and the villain has a 30 percent chance of killing the hero. Of course, the game ends if the hero is killed.

The villain will remember if the player has shot at him at any time throughout the game. The variable H is used to present this information. If H is set to a value of 1, the villian will shoot at the hero each time they meet, whether the player fires again or not.

If the villain shoots and misses, he will capture the hero in the usual manner.

Incidentally, the villain can move freely from room to room, regardless of the presence of doors. The player's movements are restricted to going through doors.

The villain is not the only obstacle in THE GREAT ESCAPE. A ghost haunts one of the rooms. See lines 7000-7140. This ornery spirit will block off all regular exists, forcing the player to use a secret passageway, adding 10 to his move count.

A gorilla (lines 7150 through 7420) may also block the movements of the hero. Ordinarily, the only way past the gorilla is to use a secret passageway, as with the ghost. However, if the player has the gun, he may try shooting the gorilla. He should bear in mind that if he misses, he'll have a very upset gorilla on his hands (lines 7260 through 7290).

There is another sure way to get past the gorilla. One of the other rooms in the building contains a bunch of bananas. If the player hs found the bananas and picked them up, he can feed them to the gorilla. The grateful ape will then leave and will not reappear for the remainder of the game.

The hero can eat the bananas himself at any time he is carrying them. On each move he will be asked if he wants to eat the bananas. Ordinarily there is no advantage to eating them (other than stopping the constantly repeated question). But if the player is injured and looking for the first aid kit, eating the bananas will extend the time limit.

There is only one bunch of bananas in the maze building. Once the player or gorilla eats them (or they are given to the robot, discussed shortly), they are gone for the rest of the game.

If the player finds himself in the room with the robot (lines 5900 through 6440), a menu of options will be displayed. These include:

CRY
DISMANTLE ROBOT (only if player has screwdriver)
EAT ROBOT
FEED BANANAS TO ROBOT (only if player has bananas)

GIVE COLD COINS TO ROBOT
PUSH PAST ROBOT
SHOOT ROBOT (only if player has gun)
X—USE SECRET PASSAGEWAY

To select the desired action the player enters the first letter. For example, to "Push past robot", enter "**P**". The entire phrase can be typed in, but the program is set up to look at just the first letter.

Several of these options can be helpful, while others have no real effect. Some of them may even be dangerous.

One hint to remember is that every robot has its price but it's very dangerous to offer more than you have.

In his travels through the building, the player may also encounter an Evil Merchant (lines 6680 through 6970). When this happens, the player must either go south, or try to bargain with the merchant for passage north. The merchant will make counter offers, but he may accept less. Bargain.

The MAD GAMBLER, on the other hand, doesn't give the player any choice in the matter. See lines 6450 through 6670.

When the hero wanders into the MAD GAMBLER's room, he must join in a game of "High Card". If he wins (draws the high card), his gold coins are doubled, and he moves one room north. If he loses, his gold coins are reduced by half, and his move count increases by three.

The game continues until the player either wins, or loses often enough so that his supply of gold coins becomes too small to merit the MAD GAMBLER's interest. If the player does not have enough gold coins to play, he is thrown into room 100.

Two wicked witches also lurk within the maze building. One will rearrange several of the room doors (lines 7460 through 7580), and the other will shift the trap doors to different rooms (lines 7430 through 7450). This may work out to the player's advantage, but it is more likely to be a nuisance.

Some of the rooms are booby-trapped with trap doors, which will randomly relocate the player into the southern half of the maze. This is done in lines 10750 through 10810.

If the player happens to be in the southern most section of the building when he falls through a trap door, he may find himself moved to the north. So trap doors aren't always bad, just usually.

A snake is slithering about in one of the rooms (lines 5800 through 5890). If the player comes into this room, he will be bitten and must locate the first aid kit within 30 moves or die (lose the game).

If he happens to come back into the room with the snake and gets bitten again, he dies immediately.

If the hero is carrying the gun, he may shoot at the snake. Even if he doesn't kill it, he may frighten if off. It will reappear in a nearby room.

Another room contains a puppy dog (lines 5620 through 5750). Usually, the puppy dog will not bother the player, but it will occasionally bite. The hero must then find the first aid kit within 100 moves.

If the player has the gun, the puppy dog is more likely to bite. He will be given the option to shoot at the puppy dog.

Dame Fortune, and her daughter Miss Fortune are also passing through the maze. If the player encounters Dame Fortune (lines 10650 and 10660), his store of gold coins increases.

Running into Miss Fortune (lines 10670 through 10690), isn't quite so pleasurable. The player loses all his gold coins and is placed in room 100.

There is also a mysteriously ticking box in one of the rooms (lines 10270 through 10300). The player is asked if he wants to pick it up. Once he picks up the box, he cannot put it down.

Table 10.2 Routines and Subroutines Used in "The Great Escape" Program.

Routines

10-50	initialize
70-100	shuffle cards
110-450	preset variables and display introduction
500-630	current room display
650-1120	special tests for room occupants
1130-1270	player's move
1280-1330	run into wall
4000-4010	secret passageway
4020	move North
4030	move South
4040	move East
4050	move West
4060-4180	cut new exit
4200-4220	escape—win game
5000-5120	encounter villain
5130-5150	player dead—lose game
5160-5205	display score/end game/begin new game?
5210	kill villain
5220-5240	bomb explodes
5250-5440	Magic Lamp #1
5450-5610	Magic Lamp #2
5620-5750	puppy dog
5760-5790	wounded -- must find first aid kit message
5800-5890	snake

5900-6440	robot
6450-6670	MAD GAMBLER
6680-6970	evil merchant
7000-7140	Ghost
7150-7420	gorilla
7430-7450	wicked witch rearranges trap doors
7460-7580	wicked witch rearranges building doors
7600-7740	treasure chest

Subroutines

10000-10020	hard game preset
10030-10170	vending machine
10180-10200	villain distance display
10210-10260	shoot
10270-10300	find ticking box
10310-10340	find saw
10350-10380	find gun
10390-10410	carrying gun
10420-10440	find first aid kit
10450-10490	rub Magic Lamp?
10500-10520	find/lose map
10530-10560	find bananas
10570-10610	carrying bananas/eat?
10620-10640	battery recharger
10650-10660	Dame Fortune
10670-10690	Miss Fortune
10700-10720	Good Fairy
10730-10740	batteries dead
10750-10810	trap door
10820-10910	card display
10920-10950	catch leprechaun
10960-11010	leprechaun distance display

BATTERIES AND MAP

If the player has not escaped from the building within a reasonable number of moves, the batteries in his flash light might go dead. The doors in each room will then not be displayed and the player must blindly try each direction. If there is no door, he will run into a wall. Bumping into too many walls can be quite hazardous to his health.

The player may purchase fresh batteries from one of the vending machines scattered throughout the maze. A set of batteries cost 25 gold coins. A player is advised not to use one of these vending machines when he has fewer than 25 coins.

A battery recharger, hidden in one of the rooms, works automatically at no cost.

The player's map tells him which room he is in, but if he does not have a map, no room number will be displayed.

At the beginning of the game, the computer asks the player if he wants a hard game. If he answers "YES", he starts out without a map. There will be fewer vending machines in the building.

A player can find or lose a map randomly throughout the game. Without a map, the player must figure out his location on his own.

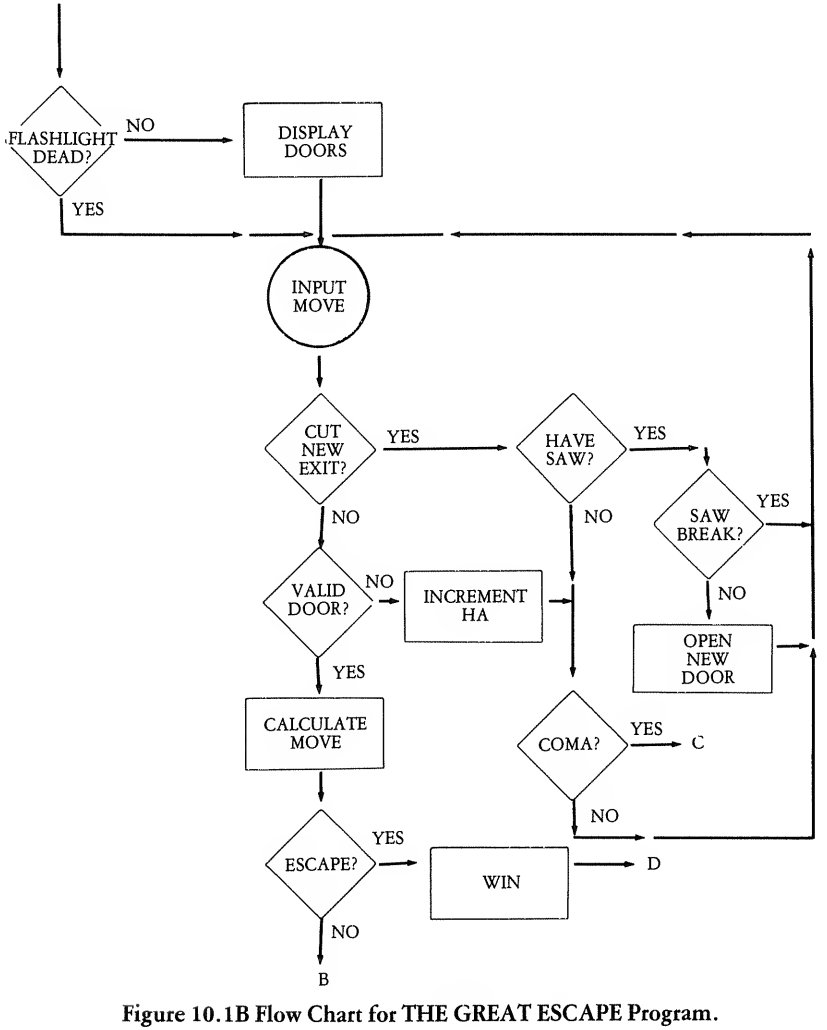


Figure 10.1B Flow Chart for THE GREAT ESCAPE Program.

Table 10.3 Variables Used in "The Great Escape" Program.

BB	ticking box location
BN	bananas
BT	battery level
BX	ticking box carried/time to explosion
CC	card counter
CO	evil merchant's counter offer
DF	Dame Fortune
GF	Good Fairy
GH	Ghost
GN	gun
GR	gorilla
G4	game counter
H	previously shot at villain?
HA	run into wall "headache" counter
K	first aid kit
K1	key
LH	have map?
LM	Magic Lamp #1
LQ	Leprechaun
L2	Magic Lamp #2
M	move counter
MC	Evil Merchant
MF	Miss Fortune
MG	MAD GAMBLER
ML	result of rubbing Magic Lamp
N	wound severity (time to find first aid kit)
O	vending machine present?
P	player's current room location
PD	puppy dog
RB	Robot
RG	Robot's price
RT	game score
RX	previous high score
S	snake
SC	screwdriver
SW	saw
TR	player's collected treasure
TX	coins in treasure chest
T5	treasure chest location
U	miscellaneous
V	villain location
VS	display villain location?
W	villain move/distance to villain

W1	Wicked Witch #1
W2	Wicked Witch #2
X	miscellaneous
X1,X2	player's card
Y,YY	miscellaneous
Y1,Y2	MAD GAMBLER's card
Z	miscellaneous

Arrays

BT(12)	vending machines
CR(52)	card values
CS(54)	card suits
E(100)	East doors
GC(100)	gold coins in rooms
N(100)	North doors
S(100)	South doors
T(10)	trap doors
W(100)	West doors

String Variables

M$	move
Q$	miscellaneous

AIDS FOR THE PLAYER

Obstacles are a necessary part of any good adventure game. However, don't stack the odds too heavily against the player. The GREAT ESCAPE includes a few helpful items and characters, such as Dame Fortune, hidden within the maze building. As you have seen, some of the obstacles may turn out to be beneficial. For instance, a wicked witch may open a door for you, or move a troublesome trap door. There are a few other helpful things the player might encounter as he attempts to escape:

A saw is hidden in one of the rooms (lines 10310 through 10340). Once the player has a saw, he can cut new exits from the rooms (see lines 4060 through 4180). For instance, if he is in a room with a door to the east, and one to the south, he can use the saw to create a new exit to the north. The new exit will remain in the room for the rest of the game (unless it is moved by a wicked witch).

When a new exit is cut, a corresponding door will not appear in the adjoining room. Ordinarily, if you pass through a door to the north, the new room will have an exit to the south. This will not be the case when a player cuts an exit for himself. The tampering of the wicked witch can also create some one-way doors.

Having the saw would make escape too easy—just cut exits to the north until you get out of the building. However, the walls are tough, and the saw might break at any time. See lines 4090 and 4150 through 4180. Once the saw is broken, it is gone for the rest of the game. This encourages the player to reserve the saw for when it is really needed.

The saw cannot be used to escape from opposing characters like the ghost, gorilla, Evil Merchant, or MAD GAMBLER.

Also hidden within the building are two Magic Lamps. They will not both appear in the same room.

When he finds one, the player is asked if he wants to rub it. If he says "NO", nothing happens, and the game continues normally. Rubbing a Magic Lamp causes one of several things to happen. Usually, the result will be to the player's advantage. The Magic Lamp may kill the gorilla, reduce the player's move count, or move the hero to the room with the saw (among other possibilities).

Chapter 11

Marketing Your Software

After all the hard work you've done to create an adventure game, why not let other people enjoy it too? You may be able to make a profit on your time investment.

Good game programs are always in demand, and there are a number of ways to market the software you've written. There are dozens of computer magazines published these days. Most of these regularly publish programs, or would if they could get good ones. These publications need to fill their pages, and they depend on free-lance material.

Write the editor first, rather than send in unsolicited work. Some magazines are reluctant to look at unsolicited submissions, especially from people they don't know.

Keep you letter short; a single typed page. Always type your query letter. Introduce yourself and *briefly* describe your program. If the editor likes the idea, he'll give you a go-ahead. This request is not a firm commitment to buy, or publish, the program in most cases. It means only that the editor is willing to consider it.

Don't be afraid to send in such a letter. The worst an editor will do is send you a form letter rejecting your idea.

If your idea is rejected, don't take it personally. There are many reasons that may have nothing to do with your ability, or the quality of the idea. The editor may have just purchased a similar piece, or he may be overstocked with game programs, or he might feel that your program just isn't quite right for his particular magazine. If you get a rejection slip shrug and try again.

It's always a very good idea to study back issues of the magazine before making a submission. A publication slanted heavily towards business programs probably wouldn't be interested in an adventure game, no matter how good it is. Some computer magazines don't publish programs at all.

You should also make sure your program is suitable to the machine(s) covered by the magazine. For instance, *80 Microcomputing* concentrates exclusively on the TRS-80 series of computers. They're not going to be very interested in a program written for an Apple unless you can adapt it for the TRS-80.

Book publishers are another possibility, especially if you have written several related programs. However, the market has been somewhat swamped with program books. These days a book of simple programs can be rather difficult to sell unless it has an unique slant.

A third potential market for your programs is software publishers, a few of which are listed with their addresses in Table 11.1. Check the ads in computer magazines for more names and addresses.

Some companies sell stock programs, usually on cassette tapes or floppy discs. Most software publishers work on a royalty basis. That is, the more copies of your program they are able to sell, the more money you will be paid. Typical contracts give you 10 to 20 percent of the selling price for the program. Often several programs will be placed on a single tape, or disc. In this case, the royalty money will be split up between the authors of each program.

Find a publisher that deals with programs for your brand of computer, and write them a brief letter describing your program(s), and asking what submission format they prefer. If they are interested, they will give you instructions.

Some software publishers are not particularly interested in submissions by amateurs. If you are not sure, query anyway. They can only say "No".

Generally, you will be asked to supply some kind of documentation, or instruction booklet for your program. While this doesn't have to be a masterpiece of literature, it is vitally important. Take your time, and make it as good as you can. Make sure you've covered everything the user needs to know thoroughly and clearly. You won't be standing at the customer's side to give instructions.

Some independent programmers take out ads in the computer magazines and try to sell their programs directly themselves. While this has proven quite profitable in a number of cases, more often it results in a major disaster. For one thing, the ads are expensive, and you have to sell quite a few copies to make the enterprise worthwhile.

Unless you are familar with the intricacies of setting up a small business and dealing with mail order, it is probably best to avoid this approach.

Computer games are extremely popular these days, and they show every sign of retaining their popularity in the near future. If you've written a

good game program, you may be able to use it as a nice supplement to your income. While you probably won't be able to make a living at it, it can be a very profitable hobby.

Have fun.

Table 11.1 Selected Software Publishers.

Adventure International
507 East Street
P.O. Box 3435
Longwood, FL 32750

Arcsoft Publishers
P.O. Box 132
Woodsboro, MD 21798

Broderbund Software Inc.
Entertainment Software Division
1938 4th Street
San Rafael, CA 94901

Datamost
9748 Cozycroft Avenue
Chatsworth, CA 91311

EPYX
P.O. Box 4247
Mountain View, CA 94040

Instant Software
Peterborough, NH 03458

K-Byte
1705 Austin Street
Troy, MI 48084

Quality Software
Suite 105
6660 Reseda Boulevard
Reseda, CA 91335

Sirius Software Inc.
10364 Rockingham Drive
Sacramento, CA 95827

Golden Flutes and Great Escapes Program Diskette for the TRS-80 Computer

LOADING INSTRUCTIONS

What You Need to Run Adventure Games for the TRS-80 Model III

Radio Shack Model III
48K memory
One disk drive
TRSDOS diskette
Standard monitor or TV with transfer switch for display

Preliminary Steps

The instructions for loading and running the programs on your TRS-80 are divided into two sections: Section 1 covers a TRS-80 Model III with two disk drives, and Section 2 covers a Model III with one drive.

Two of the programs will run with 16K of memory, but the others require 48K. Check the contents in the book to locate information on the required memory for each program. The book also contains instructions for using each of the programs.

The Overall Plan

For the Model III, you should have one diskette with the programs on it. The book provides program descriptions, and instructions for running and using the programs. The diskette with TRSDOS (Disk Operating System) must always be placed in Drive 0. Refer to your owner's manual for the proper way to insert a diskette. NEVER remove a diskette from the drive while the red light is still on, as this may cause damage to the diskette.

It is a good idea to back up your program diskette. See your owner's manual for instructions on How to Create a Backup Disk.

If you insert the program diskette and get the following message on the screen:

NS

that means the diskette you inserted is not a System diskette. You must always insert a TRSDOS diskette first. To correct the problem, take out the program diskette and insert a TRSDOS diskette. Press the reset button. Copy the programs on a TRSDOS diskette.

SECTION 1: MODEL III WITH DUAL DISK DRIVES

Start Up Steps

To load and run the programs, follow these steps:

1. Turn on the computer. The red light on the drives flashes on and off.
2. When the red light goes off, insert a TRSDOS diskette in drive 0 (bottom drive).
3. Place the program diskette in drive 1 (upper drive).
4. Press the orange reset button (upper right hand corner). A message with a description of your system and the copyright information is displayed. You will see

   Enter Date (MM/DD/YY)?

5. Be sure to enter two digits each for the month, date, and year (i.e., 09/03/82), and press ENTER key. Then see

   Enter Time (HH:MM:SS)?

6. Enter hour, minute, and seconds, also in two digits (11:05:57). If you don't need the time, go ahead and press the ENTER key (next to the bottom row, right side). The next message is

   TRSDOS Ready

7. To get a listing of the program names on the diskette, type

   DIR :1

   and press the ENTER key. The names of the programs appear on the screen. Look at the names of the programs.

   GOLD/BAS
   ESC/BAS

MARS/BAS
HUNT/BAS

Notice how each program title is followed by a slash (/) and the letters BAS. This indicates that the programs are written in BASIC. The /BAS must follow the name of each program when loading the program.

8. Type

   BASIC

   and press ENTER. The screen displays

   How many files?

9. It is not necessary to answer this question, so press ENTER. The next display is

   Memory Size?

10. This question does not need an answer either, so press ENTER again. Once again, information regarding your system and copyright appears. The screen displays

    Ready

11. You are ready to load a program. Suppose you want to load the program called MARS. Type

    LOAD "MARS/BAS"

    Notice that the name of the program is surrounded by quotation marks (" "). Remember that /BAS is included in the program name. The quotation marks, the slash, and the BAS must be included when you type the program name. If any one of them is left out, an error will result and the message

    Files Not Found

    will be displayed. If this happens, check to make sure there are quotation marks at the beginning and end of the program name, that the title is spelled correctly, that /BAS is included within the quotation marks and that there are no blank spaces in the program name. Retype the program name and press ENTER. If the program name is entered correctly, the screen displays

    Ready

12. To use the program, type

    RUN

    and press ENTER. The program is ready to go. If at any time you want to stop a program, press the BREAK key. You can load and run the same program or a different one by going through steps 11 and 12 again.

SECTION 2: MODEL III WITH A SINGLE DISK DRIVE

Start Up Steps

Use a TRSDOS diskette to make a copy of the program diskette. Follow these steps to backup, load and run the programs:

1. Turn on the computer. The red light on the drive comes on.
2. Have a formatted TRSDOS diskette ready to use. See your owner's manual for instructions on how to make a backup copy of a TRSDOS diskette. When the light on the drive goes off, insert the program diskette.
3. Press the orange reset button. A short message appears briefly on the screen about the use of the Transfer Utility Program, followed by the list of program names.

   After the program name, there is a slash (/) and the letters BAS. BAS means that the program is written in BASIC. Next there is a number (i.e., 04, 02, 05) and the abbreviation Grans. The abbreviation stands for granual which is a unit of measurement on a diskette (refer to your owner's manual for detailed information). Each program name must include the name the slash, and the BAS.

   At the bottom of the screen, this message appears:

   XX files, XXX to be copied.
   Destination Drive?

   This means there are XX files that take up XXX granuals of space on the diskette. Your TRSDOS diskette must also have at least this amount of free granuals for all of the files to be copied.

4. In respons to

   Destination Drive?

   type

   0

and press the ENTER key. A new line flashes at you from the bottom of the screen:

Mount DESTINATION disk (key)

The DESTINATION disk is your TRSDOS diskette. The SOURCE disk is the one with the programs on it. KEY means press the ENTER key. Take out the diskette in the drive (this should be the SOURCE disk).

5. Insert the DESTINATION diskette (the one with TRSDOS on it). Press ENTER. The screen then displays something similar to

Disk Name/Date:
TRSDOS-09/09/82
XXX Gran available on destination disk

The computer reads your disk and reports the name. Remember when you formatted and copied the TRSDOS diskette and were asked for the date? Well, this is the date reported here. The computer also checks to make sure there is enough room on the diskette to copy the files.

All programs are copied automatically. A flashing message appears at the bottom of the screen:

(KEY) to proceed with copy.

6. (KEY) means to press the ENTER key. With the DESTINATION disk (TRSDOS) in the drive, press ENTER. The drive light comes on and the screen displays

Copy File = =))GOLD/BAS/21 Grans

The last message flashes. As you have probably noticed, any time you are to change diskettes, the message flashes. The Copy File tells you what program is being written to the DESTINATION disk and how many granuals it requires.

7. Insert the SOURCE diskette and press ENTER. The next file is read from the diskette and another flashing message appears:

Mount DESTINATION disk (KEY)
Remove SOURCE disk

8. Insert the DESTINATION disk. The last file read from the SOURCE diskette (HUNT/BAS/09 GRANS) is written on the DESTINATION disk.

And so the process goes. Remove the DESTINATION disk and insert the SOURCE disk until all files have been copied. This usually takes about 5 minutes. When the last program is written on the DESTINATION disk, this message appears:

Updating Destination Directory
Mount Destination disk (KEY)

9. The DESTINATION disk is the one already in the drive, so press ENTER. The screen displays:

Copy Completed
(KEY) to Boot

10. Press ENTER to "boot", or load the operating system into the computer.

At this point, you have a master copy of the programs without TRSDOS, and a copy with TRSDOS. Safely store the master copy of your diskette that contains TRSDOS and the programs. To load and run your programs, follow steps 7-12 from SECTION 1.

There is no warranty of representation by dilithium Press or the authors that these programs will enable the user to achieve any particular result.

Suite 151
8285 S.W. Nimbus
Beaverton, OR 97005
(503) 646-2713 or
toll free (800) 547-1842

INDEX

TRS-80 USERS TAKE NOTE . . .

If you like *GOLDEN FLUTES AND GREAT ESCAPES How to Write Your Own Adventure Games,* you'll appreciate having the programs on a disk or cassette which is ready to run on your TRS-80 Model III or Model 4 computer. The software has a "forever guarantee" (any problems, simply return the disk or cassette with $5 and we'll send you a new one). Not only will it save your typing time, the software will save you time fretting about errors that are so easy to make.

Interested?

☐ You bet I'm interested in software for GOLDEN FLUTES AND GREAT ESCAPES!
Please send me a
___5¼" disk (0-0031-1010E)
___cassette (0-0031-1010I) for my TRS-80.

☐ Please find my check in the amount of $24.95 and rush my TRS-80 software to the address below.

☐ Please charge my VISA___M/C___ and send my TRS-80 software to the address below.

Acct #_____ Exp. Date _____

Signature _____

Name _____

Address _____

City, State, Zip _____

(To expedite your order, phone 1-800-547-1842 and charge to your VISA or M/C)

☐ Please send me your catalog entitled Brain Food™.

DILITHIUM PRESS BOOKS AND BOOK/SOFTWARE PACKAGES ARE ALSO AVAILABLE AT YOUR LOCAL BOOKSTORE AND COMPUTER STORE.

PLACE
STAMP
HERE

dilithium Software
P.O. Box 606
Beaverton, OR 97075

You move cautiously down a long, dark corridor beneath the castle of Nembuzur. You notice that your torch is beginning to burn a little low. You'll have to find a new one soon. At the end of the hall you come to three closed doors. After a moment's hesitation, you open the third door. A saber-toothed tiger leaps out at you and gashes your left arm. Quickly you draw your magic sword . . . .

Have you ever imagined an exciting adventure game? Would you like to learn how to program it on your computer? This book:

- Gives you the basic rules of designing original interactive adventure games.
- Describes the process of creating your own programs.
- Makes programming in BASIC easier by giving you special programming tricks.
- Includes 4 complete adventure game programs with full explanations of how everything works.

PROGRAMS INCLUDED: Golden Flutes • Great Escapes • Mars • Treasure Hunt

You can either type in the programs yourself or buy the book and software as a package. Book/Software packages include a 5¼" disk or cassette containing all of the programs, the loading instructions and a warranty card with our "Forever Replacement Guarantee."

Software is available for TRS-80 Model III and Model 4 computers with: *48K memory, 1 disk drive or cassette recorder, and monitor or television.*

dilithium Software *offers technical support over the telephone. With our toll-free number and friendly staff, we can answer all your questions about dilithium Software. Outside of Oregon dial* **(800) 547-1842**, *in Oregon call 646-2713.*

0 56089
45078 00995
ISBN 0-88056-089-4
>>$9.95

dilithium Press
SOFTWARE